

# **Explicit Substitution and Sorted Bigraphs**

A dissertation submitted to the  
**University of Dublin, Trinity College**  
for the degree of  
**Doctor of Philosophy**

**Shane Thomas Ó Conchúir**

Department of Computer Science  
University of Dublin, Trinity College  
Ireland

November 2008

## Declaration

This thesis has not been submitted as an exercise for a degree at any other university. Except where otherwise stated the work described herein has been carried out by the author alone.

This thesis may be borrowed or copied upon request with the permission of the Librarian, University of Dublin, Trinity College. The copyright belongs jointly to the University of Dublin and the author.

Signed:

---

Shane Thomas Ó Conchúir

November 19, 2008



# Summary

This dissertation investigates the notions of sorting and explicit substitution in the framework of bigraphs. We present kind sortings which endeavour to add enough structure to bigraphs to allow faithful representations, within this framework, of both the nested grammar of models of formal calculi and the hierarchical structures encountered in context-aware systems and network topographies. We also explore Milner’s bigraphical representation of explicit substitutions.

Pure, or unsorted, bigraphs are composed of a linking structure and a nesting structure, both self-reconfigurable, which respectively model mobile connectivity and mobile locality. They have a rich dynamic theory but limited applications. Sortings allow bigraphs to model more significant applications. We begin by defining kind sortings which allow us to refine the nesting structure with respect to a containment relation. We proceed by enriching the nesting structure with capacities and then proving that the sorting retains the rich dynamic theory of pure bigraphs.

Applications of the sorting include: a better modelling of formal calculi with a sorted, ordered grammar with capacities *e.g.* the  $\lambda$ -calculus; an interpretation of basic types as sets of controls and of a type preorder as subset inclusion, allowing us to present bigraphical models of typed  $\lambda$ -calculi; the introduction of basic flow control for the reaction relation, allowing us to express simple algorithms; and an encoding of semi-structured data with an ordered tree structure and capacities *e.g.* XML data and contexts.

Milner’s bigraphical encoding of his  $\lambda$ -calculus of explicit substitutions is indirectly studied. We contrast its notion of non-local substitution with the local substitution found in most explicit substitution (ES) calculi. We investigate properties of Milner’s calculus and discover that it satisfies many desirable properties of ES calculi. The set of properties it satisfies *e.g.* open confluence, full composition of substitutions, and preservation of strong normalisation, holds for few of these calculi. We attribute its properties to the non-local method of substitution which arises naturally from the bigraphical model. We present proofs of normalisation based on a novel simulation of non-local substitution with local substitution and characterise the set of strongly normalising terms of the calculus using an intersection type system.

Finally, we refine Milner’s encoding and present models of the untyped, simply typed, and intersection typed  $\lambda$ -calculi which have close static and dynamic correspondences with type derivations in the explicit substitution calculus. We use the results of our study of Milner’s calculus to reason about normalisation and confluence in these models, the last of which aims to model exactly the parallel compositions of terminating  $\lambda$ -terms.

## Explicit Substitution and Sorted Bigraphs

Approved by  
Dissertation Committee:

---

---

---

---

---

For Dad,

Who taught me that all you really need is love

# Contents

|                                             |            |
|---------------------------------------------|------------|
| <b>Summary</b>                              | <b>iii</b> |
| <b>Acknowledgements</b>                     | <b>xix</b> |
| <b>Chapter 1 Introduction</b>               | <b>1</b>   |
| 1.1 Contributions of the thesis . . . . .   | 8          |
| 1.2 Hypotheses . . . . .                    | 9          |
| 1.3 Structure of the dissertation . . . . . | 10         |
| <b>Chapter 2 Concepts and Terminology</b>   | <b>12</b>  |
| 2.1 Preliminary notation . . . . .          | 13         |
| 2.2 Category Theory . . . . .               | 14         |
| 2.3 Pure bigraphs . . . . .                 | 21         |
| 2.3.1 Place graphs . . . . .                | 25         |
| 2.3.2 Link graphs . . . . .                 | 26         |
| 2.3.3 Bigraphs . . . . .                    | 27         |
| 2.3.4 Related work . . . . .                | 32         |
| 2.4 Term rewriting systems . . . . .        | 32         |
| 2.5 The $\lambda$ -calculus . . . . .       | 35         |
| 2.6 Explicit substitutions . . . . .        | 36         |
| 2.7 Type Systems . . . . .                  | 39         |
| <b>I Kind Sorted Bigraphs</b>               | <b>42</b>  |
| <b>Chapter 3 Kind Bigraphs</b>              | <b>44</b>  |
| 3.1 Introduction . . . . .                  | 44         |
| 3.1.1 Motivation . . . . .                  | 45         |

|                                    |                                                                 |            |
|------------------------------------|-----------------------------------------------------------------|------------|
| 3.2                                | Place sorting . . . . .                                         | 50         |
| 3.3                                | Fundamental kind sorting . . . . .                              | 51         |
| 3.4                                | Kind sorting with visibility . . . . .                          | 54         |
| 3.5                                | Kind sortings with maximum capacities . . . . .                 | 57         |
| 3.6                                | Kind sorting with min-max capacities . . . . .                  | 63         |
| 3.6.1                              | Problems with kind sorting with capacities . . . . .            | 67         |
| 3.7                                | Conclusions . . . . .                                           | 68         |
| 3.8                                | Further work . . . . .                                          | 69         |
| <b>Chapter 4 Static Theory</b>     |                                                                 | <b>71</b>  |
| 4.1                                | Basic properties . . . . .                                      | 71         |
| 4.1.1                              | Hard bigraphs . . . . .                                         | 73         |
| 4.2                                | Further properties . . . . .                                    | 75         |
| 4.2.1                              | Operations and decompositions . . . . .                         | 78         |
| 4.3                                | Subcategories of kind sortings . . . . .                        | 80         |
| 4.4                                | Conclusions . . . . .                                           | 85         |
| 4.5                                | Related work . . . . .                                          | 86         |
| <b>Chapter 5 Transition Theory</b> |                                                                 | <b>87</b>  |
| 5.1                                | Transfer of dynamic theory . . . . .                            | 88         |
| 5.2                                | Basic properties . . . . .                                      | 94         |
| 5.3                                | Relative pushout creation . . . . .                             | 95         |
| 5.3.1                              | Relative pushouts for kind sortings . . . . .                   | 96         |
| 5.3.2                              | Relative pushouts for subcategories of kind sortings . . . . .  | 97         |
| 5.4                                | Pushout reflection . . . . .                                    | 99         |
| 5.4.1                              | Pushout reflection for kind sortings . . . . .                  | 99         |
| 5.4.2                              | Pushout reflection for subcategories of kind sortings . . . . . | 99         |
| 5.5                                | Conclusions . . . . .                                           | 100        |
| 5.5.1                              | Related work . . . . .                                          | 103        |
| <b>Chapter 6 Further Sortings</b>  |                                                                 | <b>104</b> |
| 6.1                                | Place sortings . . . . .                                        | 105        |
| 6.1.1                              | Deep kind sorting . . . . .                                     | 105        |
| 6.1.2                              | Deep sorting with capacities . . . . .                          | 107        |
| 6.2                                | Link sortings . . . . .                                         | 108        |



|                  |                                                                       |            |
|------------------|-----------------------------------------------------------------------|------------|
| 6.2.1            | Definitions . . . . .                                                 | 109        |
| 6.2.2            | Tiled link sorting . . . . .                                          | 110        |
| 6.2.3            | Plain sorting . . . . .                                               | 112        |
| 6.3              | Compound sortings . . . . .                                           | 115        |
| 6.3.1            | Combining sortings . . . . .                                          | 115        |
| 6.3.2            | Kind rigid control-sorting with visibility . . . . .                  | 117        |
| 6.4              | Summary . . . . .                                                     | 119        |
| 6.5              | Related work . . . . .                                                | 119        |
| <b>II</b>        | <b>Milner's <math>\Lambda_{\text{sub}}</math> Calculus</b>            | <b>122</b> |
| <b>Chapter 7</b> | <b><math>\lambda</math>-calculi with Explicit Substitution</b>        | <b>124</b> |
| 7.1              | Introduction . . . . .                                                | 124        |
| 7.2              | The calculi under discussion . . . . .                                | 125        |
| 7.2.1            | The $\lambda_{\text{xgc}}$ calculus . . . . .                         | 125        |
| 7.2.2            | $\lambda_{\text{lrx}}$ . . . . .                                      | 126        |
| 7.2.3            | $\lambda_{\text{es}}$ . . . . .                                       | 130        |
| 7.2.4            | $\Lambda_{\text{sub}}$ . . . . .                                      | 131        |
| 7.2.5            | Comparing $\Lambda_{\text{sub}}$ and $\lambda_{\text{xgc}}$ . . . . . | 132        |
| 7.2.6            | Comparing $\Lambda_{\text{sub}}$ and $\lambda_{\text{lrx}}$ . . . . . | 132        |
| 7.2.7            | Comparing $\Lambda_{\text{sub}}$ and $\lambda_{\text{es}}$ . . . . .  | 133        |
| 7.3              | PSN and composition of substitutions . . . . .                        | 133        |
| 7.3.1            | Weak/full composition . . . . .                                       | 134        |
| 7.3.2            | Breaking PSN . . . . .                                                | 134        |
| 7.3.3            | $\lambda_{\text{x}}\ c$ . . . . .                                     | 135        |
| 7.3.4            | $\lambda_{\text{xc}}$ . . . . .                                       | 136        |
| 7.3.5            | $\lambda_{\text{xc}}^-$ . . . . .                                     | 137        |
| 7.3.6            | $\lambda\sigma$ . . . . .                                             | 137        |
| 7.3.7            | $\Lambda_{\text{sub}}$ . . . . .                                      | 137        |
| 7.4              | Summary and related work . . . . .                                    | 138        |
| <b>Chapter 8</b> | <b>Inductive Proofs</b>                                               | <b>139</b> |
| 8.1              | Proof of confluence for $\Lambda_{\text{sub}}$ . . . . .              | 139        |
| 8.2              | Summary and related work . . . . .                                    | 145        |

|                                                                                                |                |
|------------------------------------------------------------------------------------------------|----------------|
| <b>Chapter 9 Strong Normalisation</b>                                                          | <b>147</b>     |
| 9.1 Simulating non-local substitution with local substitution . . . . .                        | 148            |
| 9.1.1 The problems . . . . .                                                                   | 148            |
| 9.1.2 A solution . . . . .                                                                     | 149            |
| 9.1.3 First approach: altering the target calculus . . . . .                                   | 151            |
| 9.1.4 Second approach: a pessimistic translation . . . . .                                     | 152            |
| 9.1.5 Comparison between the approaches . . . . .                                              | 152            |
| 9.2 PSN of $\Lambda_{\text{sub}}$ via simulation in $\lambda\text{blxr}$ . . . . .             | 153            |
| 9.2.1 The encoding of $\Lambda_{\text{sub}}$ in $\lambda\text{blxr}$ . . . . .                 | 153            |
| 9.2.2 The propagation calculus for the simulation . . . . .                                    | 155            |
| 9.2.3 Proof of PSN using $\lambda\text{blxr}$ . . . . .                                        | 158            |
| 9.2.4 Sketch of proof of PSN by translation to $\Lambda_I$ . . . . .                           | 160            |
| 9.3 Simulating $\Lambda_{\text{sub}}$ reduction in $\lambda\text{es}$ . . . . .                | 161            |
| 9.4 Characterisation of $\text{SN}_{\Lambda_{\text{sub}}}$ and a second proof of PSN . . . . . | 164            |
| 9.4.1 Intuitions . . . . .                                                                     | 164            |
| 9.4.2 Proof outline . . . . .                                                                  | 169            |
| 9.5 Conclusions . . . . .                                                                      | 174            |
| 9.6 Related work . . . . .                                                                     | 175            |
| <br><b>III Applications of Kind Sortings</b>                                                   | <br><b>179</b> |
| <b>Chapter 10 Expressiveness of Kind Sorting</b>                                               | <b>181</b>     |
| 10.1 Notation . . . . .                                                                        | 182            |
| 10.2 In the absence of control... . . . .                                                      | 182            |
| 10.2.1 Multiple readers . . . . .                                                              | 182            |
| 10.2.2 Smart buildings . . . . .                                                               | 183            |
| 10.2.3 Automatic teller machine . . . . .                                                      | 184            |
| 10.2.4 Draughts . . . . .                                                                      | 185            |
| 10.2.5 Boolean algebra . . . . .                                                               | 188            |
| 10.2.6 Ain't Got No/I Got Tea . . . . .                                                        | 190            |
| 10.3 ...be determined . . . . .                                                                | 191            |
| 10.4 Context-aware systems . . . . .                                                           | 193            |
| 10.5 Algorithms . . . . .                                                                      | 195            |

|                   |                                                                     |            |
|-------------------|---------------------------------------------------------------------|------------|
| 10.5.1            | “Find them and kill them” . . . . .                                 | 195        |
| 10.5.2            | Sudoku . . . . .                                                    | 200        |
| 10.6              | Semi-structured data . . . . .                                      | 203        |
| 10.6.1            | XML documents . . . . .                                             | 204        |
| 10.6.2            | BibTeX files . . . . .                                              | 204        |
| 10.7              | Conclusions . . . . .                                               | 207        |
| 10.7.1            | Related and further work . . . . .                                  | 207        |
| <b>Chapter 11</b> | <b>Models of <math>\lambda</math>-calculi</b>                       | <b>208</b> |
| 11.1              | $\lambda$ BIG . . . . .                                             | 210        |
| 11.1.1            | The encoding of $\Lambda_{\text{sub}}$ into $\lambda$ BIG . . . . . | 210        |
| 11.1.2            | Dynamic properties . . . . .                                        | 211        |
| 11.2              | Sorting out $\lambda$ BIG . . . . .                                 | 212        |
| 11.2.1            | The good, the bad, and the ugly . . . . .                           | 212        |
| 11.2.2            | $\lambda$ BIG $\sigma\lambda\nu\delta$ -sorting . . . . .           | 213        |
| 11.3              | Modelling the untyped $\lambda$ -calculus . . . . .                 | 215        |
| 11.3.1            | The sorting for $\lambda$ BIG <sup>ut</sup> . . . . .               | 215        |
| 11.3.2            | Static correspondence . . . . .                                     | 219        |
| 11.3.3            | Dynamic properties . . . . .                                        | 219        |
| 11.3.4            | Deficiency of the sorting . . . . .                                 | 220        |
| 11.4              | Modelling the simply typed $\lambda$ -calculus . . . . .            | 221        |
| 11.4.1            | The sorting for $\lambda$ BIG $\rightarrow$ . . . . .               | 221        |
| 11.4.2            | Static correspondence . . . . .                                     | 224        |
| 11.4.3            | Dynamic properties . . . . .                                        | 226        |
| 11.5              | Modelling the $\lambda$ -calculus with intersection types . . . . . | 229        |
| 11.5.1            | Bounded completeness of the type preorder . . . . .                 | 229        |
| 11.5.2            | The sorting for $\lambda$ BIG $^{\cap}$ . . . . .                   | 231        |
| 11.5.3            | Static correspondence . . . . .                                     | 236        |
| 11.5.4            | Dynamic properties . . . . .                                        | 237        |
| 11.6              | Conclusions . . . . .                                               | 238        |
| 11.6.1            | Further work . . . . .                                              | 239        |
| 11.6.2            | Related work . . . . .                                              | 241        |

|                                                                                          |            |
|------------------------------------------------------------------------------------------|------------|
| <b>Conclusions</b>                                                                       | <b>243</b> |
| <b>Chapter 12 Conclusions</b>                                                            | <b>245</b> |
| 12.1 Conclusions . . . . .                                                               | 245        |
| 12.2 Future work . . . . .                                                               | 246        |
| <b>Bibliography</b>                                                                      | <b>250</b> |
| <b>Chapter A Appendix for Part I</b>                                                     | <b>1</b>   |
| A.1 Pure bigraphs . . . . .                                                              | 1          |
| A.2 Homomorphic sortings . . . . .                                                       | 3          |
| A.2.1 Homomorphic sortings as kind sortings . . . . .                                    | 3          |
| A.2.2 Homomorphic sorting and simply typed $\Lambda_{\text{sub}}$ . . . . .              | 4          |
| A.2.3 Applicability of kind sortings . . . . .                                           | 5          |
| A.3 Proofs for Section 5.2 . . . . .                                                     | 7          |
| A.4 Proofs for Section 5.3 . . . . .                                                     | 10         |
| A.5 Proofs for Section 5.4 . . . . .                                                     | 17         |
| A.6 Proofs for Chapter 6 . . . . .                                                       | 19         |
| A.6.1 Tiled link sorting . . . . .                                                       | 19         |
| A.6.2 Plain sorting . . . . .                                                            | 23         |
| A.6.3 Compound sortings . . . . .                                                        | 23         |
| <b>Chapter B Appendix for Part II</b>                                                    | <b>26</b>  |
| B.1 Proofs for Section 8.1 . . . . .                                                     | 27         |
| B.2 PSN for subcalculi of $\Lambda_{\text{sub}}$ . . . . .                               | 29         |
| B.2.1 PSN for $\Lambda_{\text{sub}\downarrow\text{gc}}$ . . . . .                        | 30         |
| B.2.2 PSN for $\Lambda_{\text{subc}^\flat}$ . . . . .                                    | 32         |
| B.2.3 The problem with proving PSN for $\Lambda_{\text{sub}}$ . . . . .                  | 39         |
| B.3 Proof of PSN for $\lambda\text{blxr}$ . . . . .                                      | 43         |
| B.3.1 Proof strategy . . . . .                                                           | 44         |
| B.3.2 Simulation of $\lambda\text{blxr}$ in $\lambda I$ . . . . .                        | 44         |
| B.3.3 Encoding the $\lambda$ -calculus in $\lambda I$ and $\lambda\text{blxr}$ . . . . . | 47         |
| B.3.4 The encoding $j$ preserves strong normalisation . . . . .                          | 48         |
| B.3.5 Proof of PSN . . . . .                                                             | 52         |
| B.3.6 Simplification of the proof . . . . .                                              | 52         |

|                                                                                              |           |
|----------------------------------------------------------------------------------------------|-----------|
| B.3.7 Summary . . . . .                                                                      | 52        |
| B.4 Simulation of $\Lambda_{\text{sub}}$ reduction in $\lambda\text{blxr}$ . . . . .         | 53        |
| B.5 Simulation of $\Lambda_{\text{sub}}$ reduction in $\lambda\text{es}$ . . . . .           | 61        |
| <b>Chapter C Appendix for Part III</b>                                                       | <b>68</b> |
| C.1 Sudoku example . . . . .                                                                 | 68        |
| C.2 Proofs for $\Lambda_{\text{BIG}}$ $\sigma\lambda\nu\delta$ -sorting . . . . .            | 72        |
| C.3 Comparing $\Lambda_{\text{BIG}}$ to other encodings of the $\lambda$ -calculus . . . . . | 79        |
| C.3.1 Explicit substitutions in the $\pi$ -calculus . . . . .                                | 80        |
| C.3.2 Evaluation strategies . . . . .                                                        | 80        |
| C.3.3 Comparing $\lambda\text{lxr}$ and $\Lambda_{\text{BIG}}$ . . . . .                     | 81        |
| C.4 Proofs for Chapter 11 . . . . .                                                          | 83        |
| C.4.1 Simply typed $\Lambda_{\text{BIG}}$ . . . . .                                          | 83        |
| C.4.2 Intersection typed $\Lambda_{\text{BIG}}$ . . . . .                                    | 84        |
| C.5 Typing $\Lambda_{\text{BIG}}$ . . . . .                                                  | 86        |
| C.5.1 Thoughts on the typed models of the $\lambda$ -calculus . . . . .                      | 87        |

# List of Figures

|     |                                                                                                               |     |
|-----|---------------------------------------------------------------------------------------------------------------|-----|
| 1.1 | Reaction rule: The $\pi$ -calculus with summation . . . . .                                                   | 4   |
| 2.1 | A relative bound $(\vec{h}, h)$ for $\vec{f}$ relative to $\vec{g}$ (left) and a relative pushout (right) . . | 18  |
| 2.2 | Pushouts and relative pushouts . . . . .                                                                      | 19  |
| 2.3 | A nearly opcartesian arrow and a nearly jointly opcartesian cospan . . . . .                                  | 21  |
| 2.4 | Bigraphs, place graphs, link graphs, and composition . . . . .                                                | 23  |
| 2.5 | A bigraph reacts . . . . .                                                                                    | 24  |
| 2.6 | A simply typed discipline for the $\lambda$ -calculus . . . . .                                               | 41  |
| 2.7 | System $\text{add}_\lambda$ : An additive intersection type discipline for the $\lambda$ -calculus . . . . .  | 41  |
| 2.8 | System $\text{mul}_\lambda$ : A multiplicative intersection type discipline for the $\lambda$ -calculus . . . | 41  |
| 3.1 | Model: An abstract model of commuting workers. . . . .                                                        | 47  |
| 3.2 | Reaction rule: An agent moves between two rooms . . . . .                                                     | 49  |
| 3.3 | Reaction rule: An enemy eliminates an agent . . . . .                                                         | 50  |
| 3.4 | Containment graphs for kind signatures . . . . .                                                              | 64  |
| 5.1 | A labelled transition $L$ of an agent $a$ derived from a redex $r$ . . . . .                                  | 89  |
| 6.1 | Relating deep kind sorting to the poset $(\mathcal{P}(\mathcal{K}), \subseteq)$ . . . . .                     | 107 |
| 7.1 | Reduction rules for $\lambda\text{xgc}$ . . . . .                                                             | 126 |
| 7.2 | Congruences for $\lambda\text{lxr}$ . . . . .                                                                 | 128 |
| 7.3 | Reduction rules for $\lambda\text{lxr}$ . . . . .                                                             | 128 |
| 7.4 | Reduction rules for $\lambda\text{es}$ . . . . .                                                              | 130 |
| 7.5 | Reduction rules for $\Lambda_{\text{sub}}$ . . . . .                                                          | 131 |
| 8.1 | Relationship between properties of $\Lambda\text{x}$ terms . . . . .                                          | 145 |
| 9.1 | Reduction diagrams for simulating $\Lambda_{\text{sub}}$ in $\lambda\text{blxr}$ . . . . .                    | 159 |

|       |                                                                                                                                                   |     |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 9.2   | Simulation of $\rightarrow_{\text{bcgc}}$ -reduction in $\Lambda_I$ . . . . .                                                                     | 160 |
| 9.3   | Relating $\Lambda_{\text{sub}}$ terms with $\lambda_I$ terms . . . . .                                                                            | 160 |
| 9.4   | Reduction diagrams for simulating $\Lambda_{\text{sub}}$ in $\lambda\text{es}$ . . . . .                                                          | 164 |
| 9.5   | System $\text{add}_{\lambda\text{s}}$ : An additive intersection type discipline for $\lambda\text{es}$ and $\Lambda_{\text{sub}}$ . . . . .      | 168 |
| 9.6   | System $\text{mul}_{\lambda\text{s}}$ : A multiplicative intersection type discipline for $\lambda\text{es}$ and $\Lambda_{\text{sub}}$ . . . . . | 168 |
| 10.1  | Reaction rule: Multiple readers accessing a file . . . . .                                                                                        | 183 |
| 10.2  | Reaction rule: Lights that turn off when the last person exits the room . . . . .                                                                 | 183 |
| 10.3  | Reaction rule: The kind relation for the ATM example; The ATM returns €50 . . . . .                                                               | 184 |
| 10.4  | Reaction rule: A request for €50 is linked with notes of that value . . . . .                                                                     | 185 |
| 10.5  | The kind relation for the draughts example . . . . .                                                                                              | 185 |
| 10.6  | Reaction rule: A hop ends (all sites contain playing pieces) . . . . .                                                                            | 186 |
| 10.7  | Reaction rule: A piece is crowned . . . . .                                                                                                       | 187 |
| 10.8  | Reaction rule: A piece advances . . . . .                                                                                                         | 187 |
| 10.9  | Reaction rule: A player considers another move . . . . .                                                                                          | 187 |
| 10.10 | Reaction rule: A player takes a piece . . . . .                                                                                                   | 187 |
| 10.11 | Reaction rule: Boolean AND . . . . .                                                                                                              | 189 |
| 10.12 | Reaction rule: Boolean OR . . . . .                                                                                                               | 189 |
| 10.13 | Reaction rule: Boolean IMPLIES . . . . .                                                                                                          | 189 |
| 10.14 | Reaction rule: Boolean EQUIV . . . . .                                                                                                            | 189 |
| 10.15 | Reaction rule: Boolean NOT and a garbage collection ‘rule’ . . . . .                                                                              | 189 |
| 10.16 | The kind relation for the adventure game example . . . . .                                                                                        | 190 |
| 10.17 | Reaction rule: Taking and dropping items . . . . .                                                                                                | 190 |
| 10.18 | Reaction rule: Drop ‘no tea’ . . . . .                                                                                                            | 191 |
| 10.19 | Reaction rule: An agent moves between two rooms . . . . .                                                                                         | 192 |
| 10.20 | Reaction rule: An enemy moves between two rooms . . . . .                                                                                         | 192 |
| 10.21 | Reaction rule: An enemy eliminates an agent . . . . .                                                                                             | 192 |
| 10.22 | Reaction rule: An agent collects a dot . . . . .                                                                                                  | 192 |
| 10.23 | Kind relation for the Location-aware Printing System . . . . .                                                                                    | 195 |
| 10.24 | Multi-nodes for the “copy all devices” query . . . . .                                                                                            | 196 |
| 10.25 | Kind relation for the “copy all devices” query . . . . .                                                                                          | 197 |
| 10.26 | Reaction rules: The set of rules for the copy algorithm . . . . .                                                                                 | 198 |
| 10.27 | Example run of the flattened copy operation . . . . .                                                                                             | 199 |
| 10.28 | Reaction rule: Removing possibilities . . . . .                                                                                                   | 201 |

|       |                                                                                                        |     |
|-------|--------------------------------------------------------------------------------------------------------|-----|
| 10.29 | Reaction rule: Filling a square through inference . . . . .                                            | 202 |
| 10.30 | Distributed Sudoku . . . . .                                                                           | 203 |
| 10.31 | XML documents as ground kind sorted bigraphs . . . . .                                                 | 205 |
| 10.32 | An example of a kind relation for BibTeX files . . . . .                                               | 206 |
| 11.1  | Sorting scheme for bigraphical models of $\lambda$ -calculus . . . . .                                 | 209 |
| 11.2  | Comparing the AST of a $\lambda$ term to the corresponding bigraph . . . . .                           | 211 |
| 11.3  | The type system for untyped $\Lambda\text{BIG}$ . . . . .                                              | 215 |
| 11.4  | Two presentations of the kind sorting for $\Lambda\text{BIG}$ . . . . .                                | 216 |
| 11.5  | Ion schema for $\Lambda\text{BIG}^{\text{ut}}$ . . . . .                                               | 218 |
| 11.6  | Reaction rule: Parametric reaction rule schema for $\Lambda\text{BIG}^{\text{ut}}$ . . . . .           | 218 |
| 11.7  | Encoding of untyped $\Lambda_{\text{sub}}$ terms into $\Lambda\text{BIG}^{\text{ut}}$ . . . . .        | 219 |
| 11.8  | Typing rules for simply typed $\Lambda_{\text{sub}}$ . . . . .                                         | 221 |
| 11.9  | Representation of the simply typed $\Lambda\text{BIG}^{\rightarrow}$ sorting . . . . .                 | 223 |
| 11.10 | Ion schema for $\Lambda\text{BIG}^{\rightarrow}$ . . . . .                                             | 225 |
| 11.11 | Reaction rule: Parametric reaction rule schema for $\Lambda\text{BIG}^{\rightarrow}$ . . . . .         | 225 |
| 11.12 | Encoding of simply typed $\Lambda_{\text{sub}}$ terms into $\Lambda\text{BIG}^{\rightarrow}$ . . . . . | 226 |
| 11.13 | Reduction cubes for $\Lambda\text{BIG}^{\rightarrow}$ and $\Lambda\text{BIG}^{\cap}$ . . . . .         | 228 |
| 11.14 | An RPO, a join in $(\tau, \ll)$ , and a join in $(\mathcal{P}(\mathcal{K}), \subseteq)$ . . . . .      | 231 |
| 11.15 | Ion schema for $\Lambda\text{BIG}^{\cap}$ . . . . .                                                    | 234 |
| 11.16 | Reaction rule: Parametric reaction rule schema for $\Lambda\text{BIG}^{\cap}$ . . . . .                | 234 |
| 11.17 | Interfaces of the reaction rules . . . . .                                                             | 235 |
| 11.18 | Encoding of intersection typed $\Lambda_{\text{sub}}$ terms into $\Lambda\text{BIG}^{\cap}$ . . . . .  | 237 |
| 11.19 | Representation of the place graph sorting of $\text{Homer}\sigma$ as a kind sorting . . . . .          | 242 |
| 12.1  | Proposed reaction rule to model non-local substitution of metavariables . . . . .                      | 248 |
| A.1   | Containment conditions of homomorphic sortings and kind sortings . . . . .                             | 4   |
| A.2   | Reaction rule: The $\pi$ -calculus with summation . . . . .                                            | 6   |
| A.3   | Constructing a paired sorted RPO . . . . .                                                             | 24  |
| A.4   | Pushouts are (weakly) reflected . . . . .                                                              | 24  |
| B.1   | The relations between the $\lambda$ -calculus, $\lambda\text{blxr}$ , and $\lambda I$ . . . . .        | 45  |
| B.2   | Strong simulation of $\rightsquigarrow$ through $\mathcal{G}$ . . . . .                                | 51  |
| B.3   | Sets of free variables . . . . .                                                                       | 55  |



|      |                                                                                    |    |
|------|------------------------------------------------------------------------------------|----|
| B.4  | Abbreviations in Figure B.5 . . . . .                                              | 55 |
| B.5  | Translation for $C[(\lambda x.v)w]$ , $x$ a free variable of $v$ . . . . .         | 55 |
| B.6  | Sets of free variables . . . . .                                                   | 56 |
| B.7  | Abbreviations in Figure B.8 . . . . .                                              | 56 |
| B.8  | Translation for $C[v[x/w]]$ , $x$ a free variable of $v$ . . . . .                 | 56 |
| B.9  | Sets of free variables . . . . .                                                   | 57 |
| B.10 | Abbreviations in Figure B.11 . . . . .                                             | 57 |
| B.11 | Translation for $C[v[x/w]]$ , $x$ not a free variable of $v$ . . . . .             | 57 |
| B.12 | Sets of free variables . . . . .                                                   | 58 |
| B.13 | Abbreviations in Figure B.14 . . . . .                                             | 58 |
| B.14 | Translation for $C[v[x/w]]$ , $x$ not a free variable of $v$ . . . . .             | 58 |
| B.15 | Free variables and abbreviations for Figures B.16 and B.17 . . . . .               | 59 |
| B.16 | Translation for $C_1[C_2[x][x/v]]$ . . . . .                                       | 59 |
| B.17 | Translation for $C_1[C_2[v][x/v]]$ , $x$ a free variable of $C_2[v]$ . . . . .     | 59 |
| B.18 | Translation for $C_1[C_2[v][x/v]]$ , $x$ not a free variable of $C_2[v]$ . . . . . | 59 |
| B.19 | Free variables and abbreviations for Figures B.20 and B.21 . . . . .               | 60 |
| B.20 | Translation for $C[v[x/w]]$ , $x$ not a free variable of $v$ . . . . .             | 60 |
| B.21 | Translation for $C[v]$ . . . . .                                                   | 60 |
| B.22 | Equations for the cases where $x \notin \mathbf{fv}(D[u])$ . . . . .               | 64 |
| B.23 | Equations for the cases where $x \in \mathbf{fv}(D[u])$ . . . . .                  | 65 |
| C.1  | The initial puzzle . . . . .                                                       | 69 |
| C.2  | The initial puzzle with marked empty squares . . . . .                             | 69 |
| C.3  | Remove all marks which are not consistent . . . . .                                | 70 |
| C.4  | Finish some squares and repeatedly remove marks . . . . .                          | 70 |
| C.5  | After inferring what some squares must contain . . . . .                           | 71 |
| C.6  | The solution . . . . .                                                             | 71 |
| C.7  | Full evaluation strategy (top) and the call-by-value evaluation strategy (bottom)  | 80 |
| C.8  | Typing rules for $\lambda\text{BIG}$ . . . . .                                     | 87 |

# List of Tables

|      |                                                                                           |     |
|------|-------------------------------------------------------------------------------------------|-----|
| 5.1  | Properties of kind sortings . . . . .                                                     | 101 |
| 5.2  | Proven properties of subcategories of kind sortings with visibility . . . . .             | 102 |
| 10.1 | Truth table representing mutually exclusive reaction rules (empty rows omitted) . . . . . | 193 |
| 10.2 | The behaviour of the algorithm from the point of view of $\mathbf{f}$ . . . . .           | 199 |

# Acknowledgements

I would like to thank my supervisor, Mícheál Mac an Airchinnigh, for an inspirational, incomparable, and unforgettable undergraduate mathematics course which raised more questions than answers (a sign of a good education) and for introducing me to category theory and bigraphs.

The core of this dissertation arose from working with Martin Elsman and Thomas Hildebrandt during a stay, kindly facilitated by Lars Birkedal and Annette Hjørt Knudsen, with the Bigraphical Programming Languages group at the IT University of Copenhagen. Much of the technical work in this dissertation has its origins in that visit and conversations with the group; indeed this document may not have been written otherwise. I remain extremely grateful for the welcome I received and for the subsequent communications.

The remainder of the work presented here is largely due to the collaboration with Delia Kesner in Chapter 9 which inspired ideas in later chapters, particularly the model of  $\Lambda_{\text{sub}}$  with intersection types. I would like to thank Delia for her patience and ever-helpful comments.

Many thanks are due to Stéphane Lengrand for taking time to step through some of my technical details with me. I also wish to thank Robin Milner, Mikkel Bundgaard, Søren Debois, Kristoffer Rose, and François-Régis Sinot for their helpful correspondence and references, and others at ITU such as Rasmus Mogelberg and Troels C. Damgaard for making my stay there pleasant. This work was partially supported by the Irish Research Council for Science, Engineering, and Technology: funded by the National Development Plan.

In the earlier years of my studies, the comradery of Robert Byrne and Colin Little, the advice of Malcolm Tyrrell, and the frankness of Colman Reilly were invaluable compasses which helped steer me in better directions. My long-suffering office and flatmate Malcolm Dowse was a constant distraction what between discussing music, playing Nintendo<sup>®</sup>, and having the *occasional* pint.

For that, I will always be thankful. I must also thank the remaining members of the FMG with whom I had contact – Andrew, Glenn, Arthur, Hugh, Edsko (honorary member), Wendy, and Daniel – for listening to me trying to explain my work at them whilst providing a diversity of interesting topics themselves.

I owe personal debts of gratitude to Owen Conlan and my aunt Anne O'Connor who were always available with good advice on getting me through my doctorate.

And how could I not thank Mssrs. John Hamill, Alan Farrell, and David Hardy for providing me with endless buckets of beer, coffee, and abuse (in no particular order) over the years.

Most of all, I wish to thank Mum, Sinéad, and Madiha for their love, patience, and understanding throughout and beyond.

**Shane Thomas Ó Conchúir**

*University of Dublin, Trinity College*

*November 2008*

# Chapter 1

## Introduction

*The scene ends badly  
As you might imagine  
In a cavalcade of anger and fear  
There will be feasting and dancing  
In Jerusalem next year  
I am going to make it through this year  
If it kills me*

*This Year* – The Mountain Goats

The advent of pervasive computing has given rise to many new challenges for software engineers. Not only are we faced with the ubiquity of computation, the behaviour of these computational devices can depend on interaction and *communication* with others, either global or local. Furthermore, the *mobility* of devices, processes, and communication links – whether physically or conceptually – becomes increasingly prevalent as technology becomes increasingly integral to society. Yet another challenge lies in the *means of communication*; not only must we be able to reason about the process of communication, we must define how information is communicated *i.e.* we must consider structured data. If we are to expect certain behaviours from these systems, or some level of security, we require analytical tools which are powerful enough to describe all of these issues.

The behavioural study of communicating systems spans over thirty years, where Hoare's Com-

municating Sequential Processes (CSP) [70] and Milner’s Communicating Concurrent Systems (CCS) [99] are widely recognised as significant steps. Both theories consider the observations that can be made of a communicating system. In CSP, the behaviour of a system can be given by a set of traces, which describe the actions which have taken place. CCS considers a more refined concept of *behavioural equivalence* *i.e.* whether two devices appear to behave in the same manner to an external observer, by adopting Park’s notion of bisimilarity [125].

These *process calculi* address issues relating to inter-process communication. However, the introduction of mobile computing presents another challenge; not only can processes communicate in parallel, they can propagate through distributed systems and reconfigure their communication links. This requires more sophisticated calculi.

Milner et al.’s  $\pi$ -calculus [115] and the equivalent join calculus of Fournet and Gonthier [57] allow the communication of data or processes through named channels. This communication of channel names admits the concept of *mobile connectivity* – connections may be passed between processes. A familiar example is the passing of mobile phone connections during the hand-over protocol [103]. The fusion calculus of Parrow and Victor [126] simplifies the  $\pi$ -calculus and considers input and output as symmetric actions.

The movement of agents through space, or *mobile locality*, is a quite different form of mobility. Its importance has risen with the prevalence of both portable – perhaps context-aware – devices, which move freely through physical spaces and with mobile code propagating through virtual space. The mobile ambients of Cardelli and Gordon [32] place emphasis on hierarchical structure, modelling the movement of entire nested environments (which could represent *e.g.* processes) through physical or virtual topologies. The distributed join calculus [57, 58] extends the join calculus with the notion of location while both the distributed  $\pi$ -calculus of Sewell [139] and the Seal calculus of Vitek and Castagna [145, 35] similarly extend the  $\pi$ -calculus. The treatment of mobility differs among the calculi *e.g.* mobile ambients have subjective mobility – agents move themselves – whereas the Seal calculus considers objective mobility where agents are moved by their context, for the purposes of security.

The process calculi we mention concentrate on different aspects of mobility and all have their applications. However, they are presented quite differently. Also, while they are borne of common notions such as guarding, name-binding, process-passing, parallel composition, locality, and relocation, their means of operational analysis via observations tends to be calculus-specific; the semantics of calculi are typically given by a labelled transition system (LTS) designed, usually *ad hoc*, by a set of rules for labelled transitions. Furthermore, the derivation of a useful set of labels

from which to describe and distinguish the behaviour of processes is a non-trivial problem.

*Process frameworks* such as Milner’s action calculi [102] or Gardner and Wischik’s explicit fusions [60] address this problem by providing a common language in which to express different calculi and reason about the interactive behaviour of processes. Both frameworks are inspired by calculi with mobile connectivity. Action calculi allow various calculi to be encoded within the framework by defining a set of basic control structures and reaction rules, allowing the combination and analysis of different calculi within the same framework.

These frameworks lack two useful features. First, they do not consider the notion of locality inherent in calculi such as mobile ambients. Secondly, they do not possess a uniform treatment of labelled transitions; an LTS is defined for explicit fusions but it is provided rather than derived. Indeed, one of the aims of action calculi is to provide a general treatment of behavioural equivalences [102]. This work was continued by Milner in the framework of *bigraphs*, which addresses both of these issues.

The bigraphs of Milner, Leifer, and Jensen [105, 91, 104, 90, 74, 73, 114] have been proposed as a framework for modelling mobile computation. Bigraphs draw on ideas from the Chemical Abstract Machine [10], interaction nets [86], Nomadic Pict [147], and the process calculi and frameworks mentioned above. Bigraphs represent both mobility of connectivity as in the  $\pi$ -calculus and locality as in mobile ambients. Linkage and location are represented by two (hence ‘bi’) almost separate structures. Similar to action calculi, these structures are defined over a set of *controls*. Mobility in a system is modelled with a set of reaction rules which describe how bigraphs in the system may reconfigure themselves, allowing many calculi and mobile systems to be modelled within the one framework.

Due to work by Leifer and Milner [91, 90], bigraphs achieve a canonical treatment of labels and behavioural equivalences<sup>1</sup>. The labelled transitions of an agent are determined purely by the reaction rules and context; the contexts themselves are used as labels. However, not all contexts should be considered as labels for both practical reasons and theoretical reasons regarding observation [140]. Leifer proposes the categorical notion of colimit, through the *relative pushout* universal construction, to identify a canonical set of minimal labels for a reactive system [90].

Many calculi have been encoded in the bigraphical setting; the asynchronous  $\pi$ -calculus [74], polyadic  $\pi$ -calculus [24], and full  $\pi$ -calculus [73], the  $\lambda$ -calculus [111, 63], finite CCS [113], condition-event Petri nets [92], Higher-Order Mobile Embedded Resources (Homer) [22], mobile ambients [73], and the Fusion calculus [64]. Furthermore, strong correspondences have been shown syntactically,

---

<sup>1</sup>At the time of writing, strong and weak bisimilarity [74, 73] have been the main focus but Leifer investigates other equivalences in a broader setting [90].

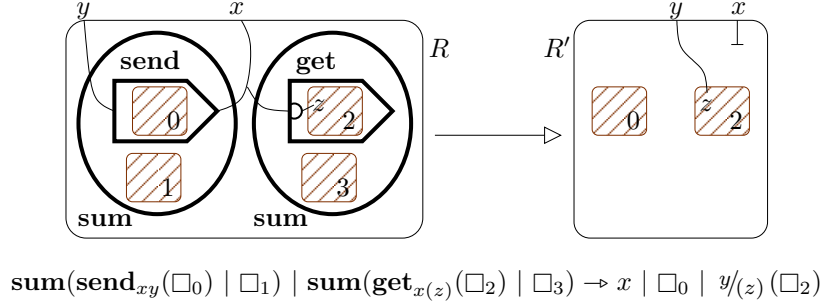


**Figure 1.1:** Reaction rule for the  $\pi$ -calculus with summation [74]

dynamically, and observationally – in terms of the original bisimilarity and that induced by the bigraphical mechanisms via relative pushouts – between the calculi and their encodings.

This is the starting point of our thesis. Bigraphs can express models of computation, communication, and mobility however as Milner states, their pure form is not likely to support significant applications [110]. Indeed, the examples of the previous paragraph all use enriched bigraphs to allow the structure of the calculus to be reflected in both the bigraphs themselves and in the labels of the LTS. We introduce an enrichment which deals with applications the existing enrichments cannot. Before we expound on this, we informally explain the bigraphical framework.

Briefly, a bigraph is analogous to a term of a process calculus or a representation of the state of some system. There are two components of a bigraph, a structure of *connections* and a structure of *locations*, both tied together with a set of *nodes*. The first component is a hypergraph and represents publicly named or private connections between nodes *e.g.* free/bound names of a calculus or access to some resource in a distributed system. The second component is a forest of trees, representing locality *e.g.* the subterm grammar of a calculus or the physical/virtual topology of some network. For example, Figure 1.1 is a visual representation of a pair of bigraphs encoding  $\pi$ -calculus contexts. The **sum**, **get**, and **send** nodes in the diagram (which respectively model summation, input, and output) are represented by labelled shapes. Locality is represented by geometric containment – the **send** node lies inside the **sum** node – and connectivity by linkage between shapes and names *e.g.* the name  $y$  is linked to an output channel which shares its name  $x$  with an input channel. The pair of bigraphs is a reaction rule representing the communication rule for a  $\pi$ -calculus with summation. The rule is *parametric*; it takes four parameters which are depicted as shaded boxes. Two of these are discarded through reaction. Besides this graphical depiction (which is quite formal), bigraphs can be formally described with a term algebra describing arrows of a category – a bigraph is an arrow of a particular type of category.



Bigraphs have quite a free structure and a strong dynamic theory but the former limits its modelling power; many models and calculi have quite specific structure and grammar requiring notions like name-binding/scoping, directed links, sorting, typing, *etc.* Enrichments, or *sortings*, of bigraphs allow these notions to be added to the framework such that if the addition is careful, it can be shown using some basic category theory that the dynamic theory is retained. To date, the published literature concentrates on enriching the linking structure to allow the modelling of calculi with mobile connectivity. Enrichments of locality have not been as widely studied: Milner and Bundgaard and Hildebrandt presented sortings for finite CCS [113] and the Homer calculus [22] respectively; Birkedal et al. describe a quite general sorting of locality (discussed in Chapter 6) [15]; and Birkedal, Debois, and Hildebrandt consider sorting of both connectivity and locality based on predicates which hold for decompositions of bigraphs. [16]

This is our contribution. We present *kind sortings* which enrich the *locality* of bigraphs while safely retaining the dynamic theory. The fundamental kind sorting is based on a suggestion by Jensen and Milner:

*Another possible refinement (of bigraphs) is a kind assigned to each node, determining the controls of the nodes it may contain. [74]*

This idea forms the basis of our work. A reactive system of bigraphs is defined over a set of *controls* representing the basic building blocks of the system. For example, a model of the  $\lambda$ -calculus may use controls for the variable, abstraction, and application constructors; a model of a smart building environment may have controls representing physical locations, mobile entities, sensors, and resources. We allow the set of controls to be enriched with a relation which describes which types of control another may parent. This allows us to model, in particular, hierarchical structures like formal grammars, models reflecting the physical world, or context-aware systems with location. We also allow the specification of *capacities e.g.* that controls may contain (at most) a certain number of particular controls.

Our motivations are as follows.

Bigraphs are intended as a step towards modelling computation on a global scale. As many practical systems undergo continuous development, Milner warns against the common practice of ‘build first, analyse later.’ He advocates that in the long-term,

*...system designs must be expressed from the outset with the concepts and notations of a theory rich enough to encompass all that the designers wish. [113]*

Bigraphs are also proposed to address the challenge of providing a uniform behavioural theory

for process calculi which respects the existing behavioural theories. Our position is that in both applications, we should strive to *internalise* concepts as much as possible: a bigraphical system representing some model should only contain bigraphs whose *structure* agrees with the model; *logical statements* should be expressed in terms of bigraphical axioms rather than some external logic; analysis of the *behaviour* of the system should be based on the entire system rather than some externally defined subsystem; and *type systems* should be represented internally. We summarise current work in these areas and our contributions in the dissertation.

**Structure** Structure is crucial to modelling context-aware and distributed systems internally as these applications have strict notions of containment and capacity in general. For example, in a model of smart buildings it would be natural to express the movement of agents inside buildings rather than the movement of buildings inside agents. Kind sortings allow bigraphs to distinguish which objects may be contained inside other objects. They also allow exact and maximum capacities to be defined in the system *e.g.* a print buffer may contain a maximum of ten pending jobs, a network contain exactly one mail server.

Published encodings of certain calculi in bigraphs are not surjective – terms of calculi are encoded into a bigraphical reactive system (Brs) where although the image of the encoding is closed under reaction, the system also contains ‘junk’ bigraphs which do not correspond to terms of the calculus<sup>2</sup>. This is true of the cited models of  $\lambda$ -calculus and Homer. In the former, the grammar of the  $\lambda$ -calculus is not respected; a  $\lambda$ -abstraction may parent more than one subterm, an explicit substitution closure may not contain a body of substitution. In both models, name-binding may be malformed. Kind sortings allow us to enforce the hierarchical structure and capacity of the grammars. We also present a separate sorting which solves most of the linking problems for the  $\lambda$ -calculus.

For technical reasons, Milner encodes the nil CCS process as a control. However in the encoding,  $0 + 0$  is not structurally equivalent to  $0$ . In terms of behaviour, this is not problematic as Milner recovers the bisimilarity  $p \mid \mathbf{nil} \sim p$  in the Brs but the structural equivalence is easily enforced if we require that each control can contain at most one nil process<sup>3</sup>.

We present kind sortings as a step towards an internal representation of locality for physical models and formal calculi in bigraphs. It subsumes the sortings of locality presented by Milner for finite CCS and by Bundgaard and Hildebrandt for Homer. Other sortings of locality, or *place*

---

<sup>2</sup>Our criteria allows certain bigraphs which do not correspond to terms of a calculus but which we do not consider to break the model *e.g.* the distribution of a  $\pi$ -calculus term over many locations or the parallel composition of  $\lambda$ -calculus terms. We interpret this as moving the original calculus into a distributed or parallel setting.

<sup>3</sup>To respect sorting, we may need to split the rule  $(\bar{x}.P + M) \mid (x.Q + N) \rightarrow P \mid Q$  on whether or not  $P = Q = 0$ .

*sortings*, have been independently studied.

The place sorting of Birkedal et al. [15] considers general relations between nodes of a bigraph and the roots of the place trees. We consider specific relations regarding containment and capacity on the more refined parent-child relation. Neither sorting generalises the other and we show in Chapter 6 that they may be safely combined.

The predicate sorting of Birkedal, Debois, and Hildebrandt subsumes many existing sortings of both locality and connectivity. It subsumes kind sortings in certain cases but in general, our sorting can not be described as a predicate sorting.

**Logic** Conforti, Macedonio, and Sassone [39] introduce a spatial logic in the style of Cardelli and Gordon [33] to bigraphs. Their logic is based on an axiomatisation of pure bigraphs and can express statements concerning both locality, connectivity, and dynamics.

Kind sortings provide a small contribution towards internal logics for reasoning about bigraphs. In kind sorted reaction rules, we can internally specify a certain type of precondition for reaction rules not allowed by pure bigraphs *e.g.* we can express reaction rules like “if a person leaves the room *and no other people remain in the room* then the light switches off.” The precondition of *absence* in the rule is our contribution. We show how sets of such rules sometimes allow us to internally enforce some flow control to the dynamics of a system.

**Semi-structured data** Cardelli observes the connection between semi-structured data and mobile computation, particularly in the description of spatial logic [28].

Kind sorting allows models of certain semi-structured data such as XML. It admits an improvement on the pure bigraphical interpretation of Conforti, Macedonio, and Sassone [39, 38] by allowing document order to be properly represented in the encoding, through an interpretation of Milner’s multi-nodes [111].

**Behaviour** The analysis of behaviour is typically based on labels of an LTS. Labels in Brss *are* bigraphs. A refinement of bigraphical structure then impacts the perceived behaviour. Informally, we want to restrict the contexts in which we distinguish behaviour; if the context is meaningless with respect to the intended model then observations based on the context are unhelpful. Keeping in line with our proposal for internalising analysis, we do not wish to ignore these contexts but rather prohibit them via sorting.

Not only do enrichments of locality constrain structure, they also focus the analysis of dynamic behaviour. This is necessary in encodings of process calculi in order that behavioural equivalences

between a calculus and its encoding correspond closely and is the common approach taken thus far in the bigraphical literature.

Enrichments of bigraphs should also preserve the dynamic theory of bigraphs where *e.g.* bisimilarity is a congruence on the transition system of minimal labels. We ensure that our kind sortings do so.

**Types** Typing is a common feature of formal calculi and can be used to specify termination, access control policies [68], or to ensure that the right type of data is sent through a communication channel.

Kind sorting allows some notion of typing to be modelled. In the dissertation, we enrich Milner’s model of the  $\lambda$ -calculus with some basic type systems. We do this by interpreting types as *sets of typed controls*. We then model *type derivations* of  $\lambda$ -terms in both the simply typed and intersection typed disciplines. We do not claim that this is the ideal way to model typed  $\lambda$ -calculi but we use it as an example of the expressiveness of the sorting.

## 1.1 Contributions of the thesis

Our primary contribution is the formalisation and investigation of Jensen and Milner’s suggestion for kind sorting, repeated above. We provide definitions for different types of kind sorting; a basic sorting, a sorting with ‘hidden local structure’, and sortings which allow us to specify exact, minimum, or maximum capacities for controls. We also investigate subcategories of these sortings, one of which recovers Milner’s sorting for finite CCS. We prove that many of these sortings and subcategories retain the dynamic theory of bigraphs and explain when the theory is broken.

Kind sorting allows us to refine the pure bigraphical model of communication and mobility. We also show that it allows us to better model the *means of communication i.e.* semi-structured data, by presenting a refinement of the encoding of XML which allows document order.

A secondary contribution is the investigation of Milner’s  $\Lambda_{\text{sub}}$  calculus.  $\Lambda_{\text{sub}}$  is an explicit substitution calculus used to present a model of the  $\lambda$ -calculus in bigraphs. However, our investigation – jointly with Delia Kesner – shows that it is not merely a means to an end.  $\Lambda_{\text{sub}}$  is interesting in its own right; it manages to satisfy properties long sought after by the explicit substitution community and only recently realised. It is remarkable that the  $\lambda$ -calculus proposed for the bigraphical model should be so closely related to the original calculus. We attribute this to its method of substitution which is *non-local*, in contrast to most explicit substitution calculi which have a local notion of substitution.

Our proofs of normalisation introduce a novel method of simulating the non-local substitution of  $\Lambda_{\text{sub}}$  with the local substitution of traditional calculi. We present two methods of simulation, the second due to Kesner. This simulation allows proofs of termination to be reflected along the simulation, allowing us to prove that if a term is  $\beta$ -strongly normalising then so is its bigraphical encoding. It also allows an simulation of reduction in  $\Lambda_{\text{sub}}$  via cut elimination in the proof nets of the multiplicative exponential fragment of linear logic (MELL) – this last result was suggested by Kesner.

This investigation fits into our main thesis.  $\Lambda_{\text{sub}}$  closely corresponds to its bigraphical encoding  $\Lambda\text{BIG}$ ; there is a strong static correspondence along the encoding and reduction in the former is matched by reaction in the latter. This dynamic correspondence means that proofs of confluence and strong normalisation in  $\Lambda_{\text{sub}}$  hold in the image of the encoding. However,  $\Lambda\text{BIG}$  contains many junk bigraphs which do not correspond to  $\Lambda_{\text{sub}}$  terms. In order to provide a faithful model of the  $\lambda$ -calculus, we set about removing these bigraphs from  $\Lambda\text{BIG}$  and almost succeed in sorting them all out. We also define a sorting scheme for modelling typed  $\lambda$ -calculi and use it to present models of both the simply typed and intersection typed  $\lambda$ -calculi. Based on our investigation of  $\Lambda_{\text{sub}}$  where we show that the intersection typing system characterises the strongly normalising terms, we conclude that the model of intersection typed  $\Lambda\text{BIG}$  is a model of parallel compositions of terminating  $\lambda$ -terms.

Lesser contributions of the thesis include: the introduction of some simple sortings for our examples; a new criteria for classifying pushout reflection which lies between the current definitions; and demonstrations of how kind sorting may be used to model flow control in the reaction relation.

## 1.2 Hypotheses

The hypotheses which we investigate in the dissertation are:

- that kind sortings retain the dynamic theory of bigraphs and that certain useful subcategories also do;
- that Milner's  $\Lambda_{\text{sub}}$  calculus and bigraphical encoding  $\Lambda\text{BIG}$  correspond closely to the  $\lambda$ -calculus in terms of simulation, confluence and normalisation properties, and typability;
- that kind sorting can be used to model semi-structured data;
- that kind sorting can be used to model basic type systems;
- that kind sorting can admit some notion of flow control to the dynamics of a system.

### 1.3 Structure of the dissertation

We resume in Chapter 2 by introducing the concepts and terminology used throughout. The remainder of the dissertation is broken into three parts.

**Part I** We introduce the definition of kind sortings and subcategories thereof and prove their hypothesised properties.

**Chapter 3** We define kind sortings which add a more refined notion of containment to bigraphs, as well as the capacity of controls. We introduce the notion of *invisible* controls which allow structure to be added to bigraphs which is not exposed to the outer interfaces. This allows an encoding of Milner’s idea of *multi-node*, which we later show allows a better bigraphical model of the  $\lambda$ -calculus and XML data.

**Chapter 4** We investigate the basic static properties of bigraphs in terms of the fundamental sorting. We concentrate on that simplest sorting as it demonstrates notions common to all the sortings. We also define *subcategories* of sortings which are later used to present encodings of simple and intersection types via the related notions of partitioned and meet subcategories.

**Chapter 5** We recall the definitions of dynamics and labelled transitions for bigraphs and as well as Leifer and Milner’s *safety* theorems which allow the dynamic theory to be preserved by certain sortings. We show that kind sortings and certain subcategories of kind sortings retain the dynamic theory using the language of obfibrations.

**Chapter 6** We present some simple, safe link sortings and discuss methods in the literature which can be used to safely combine safe sortings. We then combine kind sorting with the rigid control sorting of Birkedal et al.

**Part II** As Kesner notes, explicit substitution calculi “*are supposed to implement their underlying calculus without losing its good properties*” [78]. In this part of the dissertation, we investigate the properties of  $\Lambda_{\text{sub}}$ , Milner’s  $\lambda$ -calculus with explicit substitutions, and support our thesis that the calculus implements the untyped  $\lambda$ -calculus without losing any of the standard properties we define in the next chapter.

**Chapter 7** We introduce  $\Lambda_{\text{sub}}$  and compare it to explicit substitution calculi with local substitution.

**Chapter 8** We prove that  $\Lambda_{\text{sub}}$  can simulate  $\beta$ -reduction step-by-step and that it is confluent. We explain the problems involved with naively adapting typical proofs of preservation of strong normalisation (PSN) to  $\Lambda_{\text{sub}}$ .

**Chapter 9** This chapter contains joint work with Delia Kesner.

We present proofs of normalisation for  $\Lambda_{\text{sub}}$ . In particular we prove that  $\Lambda_{\text{sub}}$  has the property of PSN and that the strongly normalising terms are characterised by an intersection type discipline. Our proofs utilise a novel method of simulating non-local substitution with local substitution by duplicating substitutions.

Our first proof of PSN simulates reduction in a modified version of the  $\lambda_{\text{lr}}$  calculus. PSN is proven for the modified calculus by adopting Lengrand’s techniques.

We consider our second proof of PSN to be an improvement. It simulates reduction in the simpler  $\lambda_{\text{es}}$  calculus of Kesner and the encoding – due to Delia Kesner – of  $\Lambda_{\text{sub}}$  in  $\lambda_{\text{es}}$  does not require us to modify the calculus, thereby saving work reproving PSN.

We present type systems which characterise the strongly normalising terms of both  $\Lambda_{\text{sub}}$  and  $\lambda_{\text{es}}$ . They are simple extensions of intersection type systems which characterise the strongly normalising terms of the pure  $\lambda$ -calculus.

**Part III** We present applications of kind bigraphs and demonstrate their expressiveness by modelling typed  $\lambda$ -calculi and some simple algorithms.

**Chapter 10** We give informal examples of structures which kind bigraphs can model. The expressiveness gained by sorting sites of bigraphs is highlighted. It allows the preconditions of reaction rules to express the absence of exposed nodes of certain controls. We demonstrate how this may be used to model basic flow control by describing simple algorithms as Brss. We also demonstrate how kind sortings allow us to model semi-structured data with ordered locality such as XML data and BibTeX entries.

**Chapter 11** We present models of the untyped, simply typed, and intersection typed  $\lambda$ -calculus which have a close static correspondence with type derivations of the type systems and a match of reduction with reaction. We introduce a fairly complex sorting which removes most of the remaining junk bigraphs in  $\lambda\text{BIG}$  which kind sorting cannot remove. Based on results from Part II of the dissertation, we conclude confluence and normalisation properties for the models.

We present our conclusions in Chapter 12 and proofs and further discussion in the appendices.

# Chapter 2

## Concepts and Terminology

*I'll be your compass, I'll be your graph  
And your Rosetta too  
I spy, with my third eye, your hippie-dippy ingenue  
I can learn, that's all I can do . . .*

*Head For Math – Tanya Donnelly*

In this chapter, we present the fundamental concepts required for reading the dissertation. These will be unnecessary for readers familiar with the areas although our notation may differ.

After some preliminary notation, we introduce some basic category theory which is required for most of the technical work in Part I. Next we give a brief overview of the development of bigraphs and explain their structure. The following chapters will require definitions of further properties but as bigraph theory is non-trivial, we will introduce concepts as we need them. We cannot, however, reproduce the entire theory here. For the benefit of the reader, certain definitions which are assumed in the main part of the dissertation are reproduced (from Jensen and Milner's work) in the appendix.

The following sections present some fundamental rewriting and  $\lambda$ -calculus theory. Part II of the dissertation concerns explicit substitution calculi so we summarise the area and state the desirable properties of explicit substitution calculi. Finally, we discuss some simple type systems for the  $\lambda$ -calculus which we later apply to Milner's explicit substitution calculus.



## 2.1 Preliminary notation

We annotate equations with the propositions which give rise to them *e.g.* the notation

$$t \stackrel{8.3}{\equiv} u \quad \text{or} \quad t \equiv_{8.3} u$$

indicates that the congruence can be shown by an application of Lemma 8.3.

**Notation** (constant function). Given a set  $X$ , we let  $y^X$  denote the constant function  $y^X : X \rightarrow \{y\}$ . When  $y \in Y$ , we let  $y^X$  denote the function  $f : X \rightarrow Y$  where  $f(x) = y, x \in X$ .

**Notation** (inverse image). We denote the inverse image of a map  $f : X \rightarrow Y$  between two sets  $X$  and  $Y$  as  $f^{-1}$ .

**Notation** (projections). For every product  $S_1 \times \cdots \times S_n$  of  $n$  sets there are  $n$  canonical projection maps  $\pi_i : S_1 \times \cdots \times S_n \rightarrow S_i, 1 \leq i \leq n$ . We extend this notation to any subset  $T \subseteq S_1 \times \cdots \times S_n$  of an  $n$ -ary product in the obvious manner. We sometimes write  $\pi_i(P)$  meaning instead the image of the projection.

In other words, given a tuple  $c \in T \subseteq S_1 \times \cdots \times S_n$ ,  $\pi_i(c)$  denotes the  $i^{\text{th}}$  element of  $c$ .

**Notation** (powersets). We denote the powerset, the set of all subsets, of a set  $X$  as  $\mathcal{P}(X)$ . We denote the set of all finite subsets of a set  $X$  as  $\mathcal{P}_{\text{fin}}(X)$ .

**Notation** (union of disjoint sets/functions  $\uplus$ , disjoint sum of sets  $+$ ).  $S \uplus T$  denotes the union of sets  $S$  and  $T$  known or assumed to be disjoint.  $f \uplus g$  denotes the union of functions whose domains are known or assumed to be disjoint.

$S + T$  denotes the disjoint sum of  $S$  and  $T$  which disjoins the sets before taking their union. We tag the elements of the sum with an index indicating the set they came from *e.g.*  $X + P + Y$  denotes  $(\{0\} \times X) \cup (\{1\} \times P) \cup (\{2\} \times Y)$ .

**Notation** (function restriction  $\upharpoonright$ ).  $f \upharpoonright S$  denotes the restriction of a function  $f$  to the domain  $S$ .  $R \upharpoonright S$  denotes the restricted relation  $R \cap S^2$ .

**Notation** (i.h.). The abbreviation *i.h.* stands for induction hypothesis.

**Notation** ( $\stackrel{\text{def}}{=}$ ). The notation  $\stackrel{\text{def}}{=}$  means “is defined as.”

## 2.2 Category Theory

Bigraphs are presented with (and greatly benefit from) category theory. The standard mathematical text is Mac Lane's book [97]. The computer scientist may find the introductions by Awodey [7] and Barr and Wells [9] more approachable. Other recommended texts are Lawvere and Schanuel's foundational introduction to categorical reasoning and basic toposes [89] and Lawvere and Rosebrugh's proposal of categories as a foundation [88].

We will only require an understanding of some basic concepts. Our presentation is based on both Awodey's [7] and Milner's [114].

**Definition 2.1** (category). A category  $\mathbf{C}$  consists of objects  $\text{obj}(\mathbf{C})$  and arrows  $\text{arr}(\mathbf{C})$ . Each arrow  $f$  is associated with a pair of objects  $\text{dom}(f)$  and  $\text{cod}(f)$  respectively called the domain and codomain of  $f$ . The notation  $f : A \rightarrow B$  indicates that  $A$  (resp.  $B$ ) is the domain (resp. codomain) of  $f$ . For each object  $A$  of  $\mathbf{C}$ , there is an arrow  $\text{id}_A$  called the identity arrow.

The operation  $\circ$  is called composition. If  $f : A \rightarrow B$  and  $g : B \rightarrow C$  then we can form the composite  $g \circ f : A \rightarrow C$  which we sometimes abbreviate to  $gf$ .

$\mathbf{C}$  is required to satisfy the laws of associativity ( $h \circ (g \circ f) = (h \circ g) \circ f$  when  $\text{cod}(f) = \text{dom}(g)$  and  $\text{cod}(g) = \text{dom}(h)$ ) and unit ( $(f \circ \text{id}_A = f = \text{id}_B \circ f$  when  $f : A \rightarrow B$ ).

Categories are quite like directed graphs in structure.<sup>1</sup>

**Definition 2.2** (precategory). A precategory  $\mathbf{C}$  is defined exactly as a category, except that the composition of arrows is not always defined. Composition with the identities is always defined and the unit law always holds. The associativity law holds when the compositions are defined.

**Definition 2.3** (functor). A functor  $\mathcal{F} : \mathbf{C} \rightarrow \mathbf{D}$  between (pre)categories  $\mathbf{C}$  and  $\mathbf{D}$  is a mapping from objects of  $\mathbf{C}$  to objects of  $\mathbf{D}$  and arrows of  $\mathbf{C}$  to arrows of  $\mathbf{D}$  which satisfies the following laws:

- $\text{dom}(\mathcal{F}(f)) = \mathcal{F}(\text{dom}(f))$  and  $\text{cod}(\mathcal{F}(f)) = \mathcal{F}(\text{cod}(f))$ ,
- $\mathcal{F}(g \circ f) = \mathcal{F}(f) \circ \mathcal{F}(g)$ ,
- $\mathcal{F}(\text{id}_A) = \text{id}_{\mathcal{F}(A)}$ .

---

<sup>1</sup>A formal connection exists with the notion of *free category* on a graph.

These can be represented by the commuting diagrams below.

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B & \xrightarrow{g} & C \\
 \mathcal{F} \downarrow & & \mathcal{F} \downarrow & & \mathcal{F} \downarrow \\
 \mathcal{F}(A) & \xrightarrow{\mathcal{F}(f)} & \mathcal{F}(B) & \xrightarrow{\mathcal{F}(g)} & \mathcal{F}(C)
 \end{array}
 \qquad
 \begin{array}{ccc}
 A & \xrightarrow{\text{id}_A} & A \\
 \mathcal{F} \downarrow & & \mathcal{F} \downarrow \\
 \mathcal{F}(A) & \xrightarrow{\text{id}_{\mathcal{F}(A)}} & \mathcal{F}(A)
 \end{array}$$

A functor may be informally thought of as a graph homomorphism.

**Definition 2.4** (homset). *The set of arrows from  $A$  to  $B$  in a (pre)category  $\mathbf{C}$  is called a homset and is denoted as  $\text{Hom}_{\mathbf{C}}(A, B)$  or  $\mathbf{C}(A \rightarrow B)$ .*

**Definition 2.5** (faithful, full functor). *A functor  $\mathcal{F} : \mathbf{C} \rightarrow \mathbf{D}$  is faithful iff it is injective on homsets i.e. for all  $A, B \in \text{obj}(\mathbf{C})$ , the mapping  $\text{Hom}_{\mathbf{C}}(A, B) \rightarrow \text{Hom}_{\mathbf{D}}(\mathcal{F}(A), \mathcal{F}(B))$  is injective. A functor  $\mathcal{F} : \mathbf{C} \rightarrow \mathbf{D}$  is full iff it is surjective on homsets.*

**Definition 2.6** (subcategory). *A functor  $\mathcal{F} : \mathbf{C} \rightarrow \mathbf{D}$  is a sub(pre)category when it is injective on objects and homsets.  $\mathcal{F}$  defines a full subcategory exactly when it is a full functor.*

*$\mathcal{F}$  is called an inclusion functor of  $\mathbf{C}$  in  $\mathbf{D}$  and is faithful by definition.*

Informally, we can think of subcategories similarly to subgraphs.

**Definition 2.7** (tensor product, monoidal precategory). *A (strict, symmetric) monoidal precategory has a partial tensor product  $\otimes$  both on objects and on arrows. It has a unit object  $\epsilon$ , called the origin, such that  $I \otimes \epsilon = \epsilon \otimes I = I$  for all  $I$ . Given  $I \otimes J$  and  $J \otimes I$  it also has a symmetry isomorphism  $\gamma_{I,J} : I \otimes J \rightarrow J \otimes I$ . The tensor and symmetries satisfy the following equations when both sides exist:*

- (1)  $f \otimes (g \otimes h) = (f \otimes g) \otimes h$  and  $\text{id}_{\epsilon} \otimes f = f$
- (2)  $(f_1 \otimes g_1)(f_0 \otimes g_0) = (f_1 f_0) \otimes (g_1 g_0)$
- (3)  $\gamma_{I,\epsilon} = \text{id}_I$
- (4)  $\gamma_{J,I} \circ \gamma_{I,J} = \text{id}_{I \otimes J}$
- (5)  $\gamma_{I,K} \circ (f \otimes g) = (g \otimes f) \circ \gamma_{H,J}$  (for  $f : H \rightarrow I, g : J \rightarrow K$ ).

**Definition 2.8** (s-category). An s-category  $\mathcal{C}$  is a strict symmetric monoidal precategory which has:

- for each arrow  $f$ , a finite set  $|f|$  called its support, such that  $|\text{id}_I| = \emptyset$ . For  $f : I \rightarrow J$  and  $g : J \rightarrow K$  the composition  $gf$  is defined iff  $|g| \cap |f| = \emptyset$  and  $\text{dom}(g) = \text{cod}(f)$ ; then  $|gf| = |g| \uplus |f|$ . Similarly, for  $f : H \rightarrow I$  and  $g : J \rightarrow K$  with  $H \otimes J$  and  $I \otimes K$  defined, the tensor product  $f \otimes g$  is defined iff  $|f| \cap |g| = \emptyset$ ; then  $|f \otimes g| = |f| \uplus |g|$ .
- for any arrow  $f : I \rightarrow J$  and any injective map  $\rho$  whose domain includes  $|f|$ , an arrow  $\rho \bullet f : I \rightarrow J$  called a support translation of  $f$  such that:

$$\begin{array}{ll}
 (1) \quad \rho \bullet \text{id}_I = \text{id}_I & (5) \quad (\rho_1 \circ \rho_0) \bullet f = \rho_1 \bullet (\rho_0 \bullet f) \\
 (2) \quad \rho \bullet (gf) = (\rho \bullet g)(\rho \bullet f) & (6) \quad \rho \bullet f = (\rho \upharpoonright |f|) \bullet f \\
 (3) \quad \rho \bullet (f \otimes g) = \rho \bullet f \otimes \rho \bullet g & (7) \quad |\rho \bullet f| = \rho(|f|). \\
 (4) \quad \text{Id}_{|f|} \bullet f = f
 \end{array}$$

Each equation is required to hold only when both sides are defined.

**Definition 2.9** (reflecting, preserving properties). A functor  $\mathcal{F} : \mathcal{C} \rightarrow \mathcal{D}$  is said to preserve a property if, given a collection of arrows and objects of  $\mathcal{C}$  satisfying the property, the image in  $\mathcal{D}$  under  $\mathcal{F}$  of that collection also has the property.

A functor  $\mathcal{F} : \mathcal{C} \rightarrow \mathcal{D}$  is said to reflect a property if, given a collection of arrows and objects of  $\mathcal{C}$ , if its image in  $\mathcal{D}$  under  $\mathcal{F}$  satisfies the property then so does the collection in  $\mathcal{C}$ .

**Definition 2.10** (support equivalence, supported functor). Two arrows  $f, g : I \rightarrow J$  in an s-category  $\mathcal{A}$  are support-equivalent, written  $f \simeq g$ , if  $\rho \bullet f = g$  for some support translation  $\rho$ . By Definition 2.8 this is an equivalence relation. If  $\mathcal{B}$  is another s-category, then a supported functor  $\mathcal{F} : \mathcal{A} \rightarrow \mathcal{B}$  is a function on objects and arrows that preserves identities, composition, tensor product and support equivalence. If  $\mathcal{F}$  is an inclusion function then  $\mathcal{A}$  is a sub-s-category of  $\mathcal{B}$ .

**Notation.** S-categories will be accented e.g.  $\mathcal{C}$  whereas categories are denoted as  $\mathbf{C}$ . For convenience, we will write ‘subcategory’ for a subprecategory of an s-category.

**Definition 2.11** (congruence). Let  $\equiv$  be an equivalence defined on every homset of an s-category  $\mathcal{C}$ . It is said that  $\equiv$  is preserved by an operator  $*$  if  $f \equiv f'$  and  $g \equiv g'$  imply  $f * g \equiv f' * g'$  whenever the latter are defined. Then  $\equiv$  is a congruence on  $\mathcal{C}$  whenever it is preserved by composition and tensor product.

**Definition 2.12** (quotient categories). Let  $\mathcal{C}$  be an s-category, and let  $\equiv$  be a congruence on  $\mathcal{C}$  that includes support equivalence, i.e.  $\simeq \subseteq \equiv$ . Then the quotient of  $\mathcal{C}$  by  $\equiv$  is a category  $\mathbf{C} \stackrel{\text{def}}{=} \mathcal{C}/\equiv$ , whose objects are the objects of  $\mathcal{C}$  and whose arrows are equivalence classes of arrows in  $\mathcal{C}$ :

$$\mathbf{C}(I, J) \stackrel{\text{def}}{=} \{[f]_{\equiv} \mid f \in \mathcal{C}(I, J)\}.$$

In  $\mathbf{C}$ , the identities, composition and tensor product are given by

$$\begin{aligned} \text{id}_m &\stackrel{\text{def}}{=} [\text{id}_m]_{\equiv} \\ [g]_{\equiv} \circ [f]_{\equiv} &\stackrel{\text{def}}{=} [gf]_{\equiv} \\ [f]_{\equiv} \otimes [g]_{\equiv} &\stackrel{\text{def}}{=} [f \otimes g]_{\equiv} \end{aligned}$$

By assigning empty support to every arrow,  $\mathbf{C}$  may also be regarded as an s-category and  $[\cdot]_{\equiv} : \mathcal{C} \rightarrow \mathbf{C}$  is called the  $\equiv$ -quotient functor for  $\mathcal{C}$ .

In the following definition, **Ord** is the s-category of finite ordinals and functions between them.

**Definition 2.13** (wide s-category). An s-category  $\mathcal{A}$  is wide if equipped with a functor  $\text{width} : \mathcal{A} \rightarrow \mathbf{Ord}$  with  $\text{width}(\epsilon) = 0$  such that, for each bijection  $\pi$  on the ordinal  $\text{width}(I)$ , there is an isomorphism  $\pi_I : I \rightarrow I$  in  $\mathcal{A}$  with  $\text{width}(\pi_I) = \pi$ .<sup>2</sup>

The objects  $H, I, J, \dots$  of  $\mathcal{A}$  are called interfaces, and its arrows  $A, B, C, \dots$  are called contexts. The domain and codomain of a context will be called its inner and outer faces. Arrows in a homset  $\mathcal{A}(\epsilon \rightarrow I)$  – which is often abbreviated to  $\mathcal{A}(I)$  – are called ground arrows; lower case letters  $a, b, \dots$  range over these, and  $a : \epsilon \rightarrow I$  can be abbreviated to  $a : I$ .

**Definition 2.14** (span, cospan). A span is a pair of arrows with the same domain. A cospan is a pair of arrow with the same codomain.

$\vec{f}$  will be used to denote a span or a cospan of two arrows  $f_0$  and  $f_1$ . It will be made clear whether the pair is a span or cospan. We write  $f_i$  to denote one of  $\vec{f}$  and  $f_{\bar{i}}$  to denote the other.

**Definition 2.15** (bound, consistent). Given a span  $\vec{f} : H \rightarrow \vec{I}$  and a cospan  $\vec{g} : \vec{I} \rightarrow K$ , if  $g_0 f_0 = g_1 f_1$ , then it is said that  $\vec{g}$  is a bound for  $\vec{f}$ . If  $\vec{f}$  has any bound, then it is said to be consistent.

Bigraphs are arrows of specific categories. Composition in those categories is analogous to the placing of a  $\lambda$ -calculus term inside a  $\lambda$ -context. Bigraphs can also react, or reconfigure themselves,

<sup>2</sup>In place-sorted bigraphs, the isomorphisms  $\pi_I$  generally have domain  $I$  and codomain  $I'$  where  $I \neq I'$ . This is fine as both  $\text{width}(I) = \text{width}(I')$  and  $\text{width}(\pi_I) = \pi$  hold.

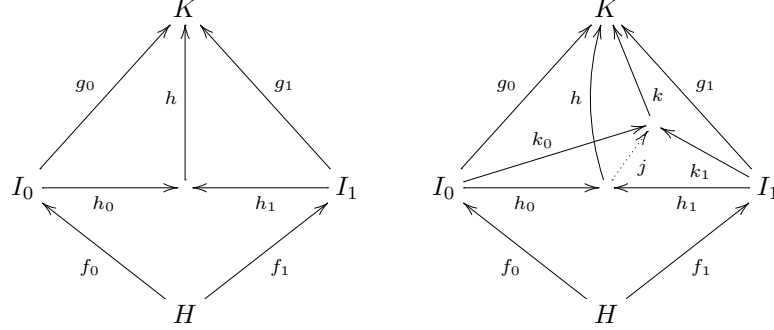


**Figure 2.1:** A relative bound  $(\vec{h}, h)$  for  $\vec{f}$  relative to  $\vec{g}$  (left) and a relative pushout (right)

according to reaction rules. This leads to a definition of labelled transition systems of bigraphs based on contexts. For example, if  $a$  placed in context  $F$  can reconfigure to  $a'$ , we can write  $a \xrightarrow{F} a'$ . However, we would not wish to take *all* contexts as labels as this would yield an intractably large relation over which to prove *e.g.* bisimilarity. Furthermore, Sewell points out that the induced behavioural equivalences are not satisfactory [140].

We want labels be to minimal in some sense – but in what *exact* sense? Leifer’s proposal [90] was to use the notion of *relative pushout* (RPO) to define minimal labels.

**Definition 2.16** (relative pushout (RPO)). *Let  $\vec{g} : \vec{I} \rightarrow K$  be a bound for  $\vec{f} : H \rightarrow \vec{I}$ . A bound for  $\vec{f}$  relative to  $\vec{g}$  is a triple  $(\vec{h}, h)$  of arrows such that  $\vec{h}$  is a bound for  $\vec{f}$  and  $hh_i = g_i (i = 0, 1)$ . The triple may be called a relative bound when  $\vec{g}$  is understood.*

*A relative pushout (RPO) for  $\vec{f}$  relative to  $\vec{g}$  is a relative bound  $(\vec{h}, h)$  such that for any other relative bound  $(\vec{k}, k)$  there is a unique arrow  $j$  for which  $jh_i = k_i (i = 0, 1)$  and  $kj = h$ .*

Specific RPOs are used as the labels of the labelled transition systems of bigraphical systems.<sup>3</sup>

**Definition 2.17** (idem pushout (IPO)). *A pair  $\vec{h} : \vec{I} \rightarrow J$  is an idem pushout (IPO) for the pair  $\vec{f} : H \rightarrow \vec{I}$  if the triple  $(\vec{h}, \text{id}_J)$  is an RPO for  $\vec{f}$  to  $\vec{h}$ .*

**Definition 2.18** (pushout). *A bound  $\vec{g} : \vec{I} \rightarrow J$  for  $\vec{f} : H \rightarrow \vec{I}$  is a pushout for  $\vec{f}$  if given any other bound  $\vec{h} : \vec{I} \rightarrow K$  for  $\vec{f}$ , there exists a unique mediator  $j : J \rightarrow K$  such that  $j \circ g_i = h_i$ .*

RPOs are pushouts in a slice category [90]. Specifically, the pushout  $(h_0, h_1, h)$  of a bound  $g_0, g_1$  with codomain  $K$  for a pair  $f_0, f_1$  is a pushout in the slice category over  $K$ . This is depicted

<sup>3</sup>However, as general constructions they are not restricted to bigraphs. Similarly, there are other presentations of bigraphs besides Milner and Jensen’s (we mention these later).

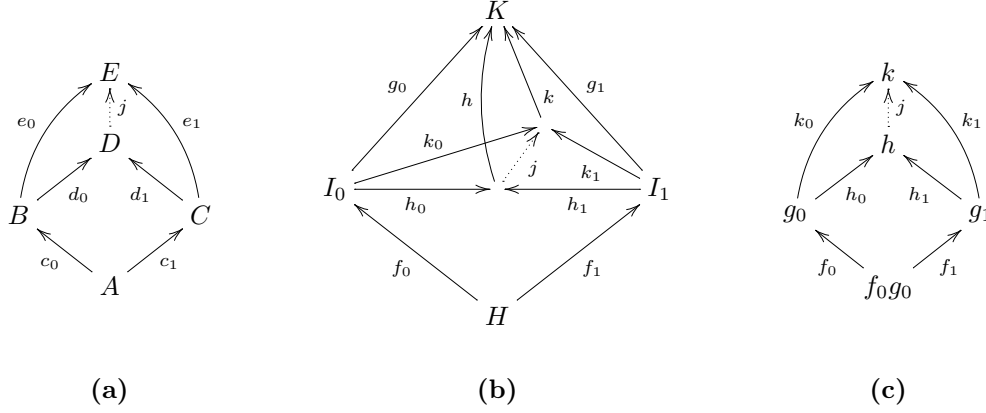


**Figure 2.2:** (a) A pushout  $(d_0, d_1)$  of  $(c_0, c_1)$  in  $\mathbf{C}$  (b) A relative pushout  $(\vec{h}, h)$  for  $\vec{f}$  relative to  $\vec{g}$  in  $\mathbf{C}$   
(c) The same relative pushout as a pushout in the slice category  $\mathbf{C} \downarrow K$

in Figure 2.2. Pushouts describe bounds which are least in some sense.<sup>4</sup> An RPO is a triple  $(h_0, h_1, h)$  such that  $h_0$  and  $h_1$  are minimal; this will hopefully become clearer once bigraphs are introduced. The intuition is that  $h_0$  and  $h_1$  are the minimal contexts such that  $h_0 \circ f_0 = h_1 \circ f_1$ ;  $h_0$  (resp.  $h_1$ ) adds the least amount to  $f_0$  (resp.  $f_1$ ) such that their composition equals  $h_1 \circ f_1$  (resp.  $h_0 \circ f_0$ ). This intuition is invaluable when trying to find proofs of RPO creation for sorted bigraphs.

The notion of relative pushout is an example of a universal construction. We do not need to present the definition of universal construction here but it is a powerful concept.

Bigraphs benefit greatly from their categorical presentation. We will investigate sorted bigraphs in the following chapters. Sorted bigraphs form their own s-categories but they are linked to pure bigraphs via *sorting functors* which are forgetful and faithful. In well-behaved sortings, much of the pure theory can be reflected along this functor.

**Definition 2.19** (sorting functor). *A sorting functor is surjective on objects and faithful.*

We adopt the terminology below for opfibrations from Jacobs and Birkedal et al. [72, 17].

Arrows or cospans with particular properties along sortings are instrumental to our proofs of RPO creation and pushout reflection. Jensen has presented conditions which are sufficient to prove that a sorting creates RPOs [73] whilst Leifer and Milner identified that if every arrow in an s-category is opcartesian with respect to a sorting functor  $\mathcal{U}$ , then this is sufficient to prove pushout

<sup>4</sup>For example, a pushouts of a pair  $f_0 : A \rightarrow B_0, f_1 : A \rightarrow B_1$  of functions of sets is a bound  $g_0, g_1$  which is the least identification on the elements of  $B_0$  and  $B_1$ .

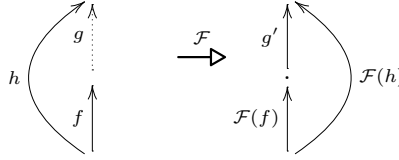
reflection along  $\mathcal{U}$  [92].

**Definition 2.20** (above, vertical, lift at an object). *Let  $\mathcal{F} : \mathbf{E} \rightarrow \mathbf{B}$  be a functor.*

*We say that an arrow  $f$  is above  $\mathcal{F}(f)$ . If  $f : I \rightarrow J$  is above an identity then we call it a vertical and denote  $f$  by  $J \uparrow I$ <sup>5</sup>.*

*Let  $E$  be an object of  $\mathbf{E}$ . An arrow of  $\mathbf{B}$  has a lift at  $E$  iff it is the  $\mathcal{F}$ -image of an arrow  $f : E \rightarrow X$ .*

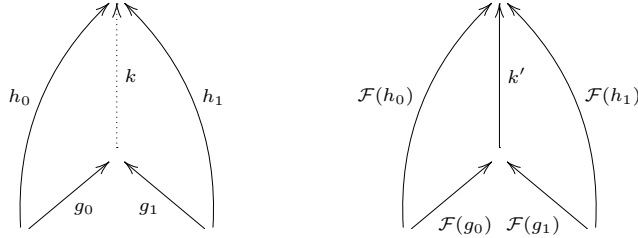
**Definition 2.21** (opcartesian). *An arrow  $f$  of a precategory  $\mathbf{D}$  is opcartesian with respect to a functor  $\mathcal{F} : \mathbf{D} \rightarrow \mathbf{C}$  iff for all  $h$  where  $\mathcal{F}(h) = g' \circ \mathcal{F}(f)$ , there exists a unique arrow  $g$  in  $\mathbf{D}$  such that  $\mathcal{F}(g) = g'$  and  $h = g \circ f$ .*



**Definition 2.22** (weak opfibration). *A functor  $\mathcal{F} : \mathbf{E} \rightarrow \mathbf{B}$  is a weak opfibration iff whenever an arrow  $f$  of  $\mathbf{B}$  has a lift at  $E$ , it has an opcartesian lift at  $E$ .*

The requirement that all arrows are opcartesian is quite strong. Birkedal, Debois, and Hildebrandt consider the notion of jointly opcartesian cospan [17].

**Definition 2.23** (jointly opcartesian cospan). *A cospan  $\vec{g}$  of a precategory  $\mathbf{D}$  is jointly opcartesian with respect to a functor  $\mathcal{F} : \mathbf{D} \rightarrow \mathbf{C}$  iff whenever  $\vec{h}$  is a cospan,  $g_0, h_0$  is a span, and  $g_1, h_1$  is a span in  $\mathbf{D}$  with  $\mathcal{F}(h_i) = k' \circ \mathcal{F}(g_i)$ , then there is a unique  $k$  in  $\mathbf{C}$  such that  $h_i = k \circ g_i$  and  $\mathcal{F}(k) = k'$ .*



**Definition 2.24** (jointly opcartesian bound (JOB)). *If a jointly opcartesian cospan  $\vec{g}$  is a bound for  $\vec{f}$ , we say that it is a jointly opcartesian bound (JOB) for  $\vec{f}$ .*

<sup>5</sup>This notation will be intuitive for our application.



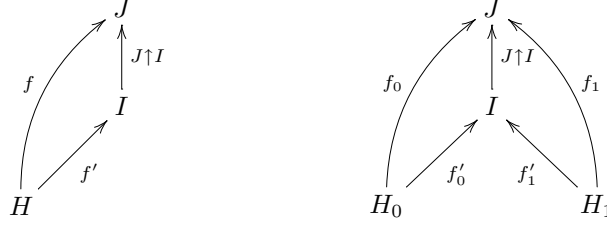


**Figure 2.3:** A nearly opcartesian arrow  $f$  (left) and a nearly jointly opcartesian cospan  $\vec{f}$  (right)

If all arrows along a functor are opcartesian then all cospans along the functor are jointly opcartesian. We will work with cospans so the property of jointly opcartesian is weaker, more generally applicable, and just as sufficient. In Chapter 5, we consider a slightly weaker property.

**Definition 2.25** (nearly opcartesian). *An arrow  $f : H \rightarrow J$  of  $\mathbf{E}$  is nearly opcartesian if there exists a vertical  $J \uparrow I$  and an opcartesian  $f' : H \rightarrow I$  such that  $f = J \uparrow I \circ f'$ .*

**Definition 2.26** (nearly jointly opcartesian). *A cospan  $\vec{f} : \vec{H} \rightarrow J$  of  $\mathbf{E}$  is nearly jointly opcartesian if there exists a vertical  $J \uparrow \hat{I}$  and a jointly opcartesian cospan  $\vec{f}' : \vec{H} \rightarrow I$  such that  $f_i = J \uparrow \hat{I} \circ f'_i, i \in \{0, 1\}$ .*

## 2.3 Pure bigraphs

As described in the introduction, bigraphs are a framework for mobile computation and communication which aims to both allow the modelling of ubiquitous computation and present a unified framework in which to analyse the behavioural equivalences of process calculi. In this section, we present some of the basic notions. The theory is quite detailed and we refer the reader to Jensen and Milner's work [74, 114] for a detailed explanation.

We adopt Milner's definitions and terminology for pure bigraphs which have been developed from work by Jensen and Milner [74, 113]. Informally, a pure bigraph is a combination of two graphs, the first of which is a partially ordered<sup>6</sup> forest of trees called a *place graph* whilst the second is a hypergraph<sup>7</sup> called a *link graph*. The place graph models hierarchies of locations. This can be used to model containment *e.g.* the intruder is in the building, the PC is inside the firewall, the  $\lambda$ -variable is under the  $\lambda$ -abstraction. The link graph instead models connectivity, connecting

<sup>6</sup>The set of roots is totally ordered, there is no order amongst children of a node.

<sup>7</sup>A generalization of an undirected graph where edges link any non-empty subset of the set of vertices.

entities in a system to each other or to identifiers *e.g.* the intruder is communicating with a foreign agent, the PC is linked to a WAN server, the  $\lambda$ -variable is named  $x$ . These two graphs are largely independent, the intuition being “*where you are doesn’t affect who you may talk to*” [109]. For example, the foreign agent or the server may be located far from the intruder or PC respectively.

A category of bigraphs is defined over a *signature*. The signature specifies the entities of the system, called *controls*. In the informal intruder example above, the controls would include intruder, building, and foreign agent. The signature also specifies whether a control may contain other controls and the number of *ports* it has to link with other controls and names.

Figure 2.4 depicts three bigraphs  $G$ ,  $F$ , and  $G \circ F$  and their constituent place graphs and link graphs. A place graph is an ordered forest of trees. The nodes at the top of the trees ( $t$  in  $G$ ,  $r_0$  and  $r_1$  in  $F$ ) are special nodes called *roots*. Some of the leaves are also special and are called *sites* *e.g.*  $r_0$  and  $r_1$  in  $G$  and  $s_0$  in  $F$ . These special nodes describe the interfaces of bigraphs, allowing us to compose two place graphs to form a larger one. In composition, one place graph is ‘planted’ inside the other by placing its roots inside the sites of the other. A link graph also involves nodes but it only cares about their ports ( $p_0, \dots, p_6$  in the diagram). The ports are linked to each other and to *inner* and *outer names*. The names of a link graph allow us to compose two link graphs to form a larger one by ‘fusing’ the links together.

A bigraph is a combination of a link graph and a place graph with the same set of nodes. This shared set  $v_0, \dots, v_5$  is the only overlap in the structures which are otherwise independent. The place graph is represented by a nesting of nodes, roots are represented by outer rectangles called *regions*, and sites are represented by shaded *holes*. The inner names of the link graph are placed at the bottom of the bigraph whereas the outer names sit up top. The bigraph  $G$  with two sites and inner names  $y$  and  $z$  may be composed with a bigraph with two roots and outer names  $y$  and  $z$ . The composition  $G \circ F$  plants  $F$  inside  $G$ , fuses the links, and forgets the places and names at the shared interface.

Bigraphs have an intuitive graphical representation but they are formal structures with a term algebra. A bigraph is an arrow of a wide s-category over some signature. The objects of this category are simple; an object is a pair  $\langle m, X \rangle$  called an *interface*, consisting of a *width*  $m$  – a finite ordinal  $\{0, \dots, m-1\}$  – and a finite set of *names*  $X$ , where the names of  $X$  are taken from an infinite set  $\mathcal{X}$ . The domain (resp. codomain) of a bigraph is called its inner (resp. outer) interface. The sites (resp. roots) of a bigraph correspond to the width of its inner (resp. outer) interface and its inner names (resp. outer names) correspond to the names of its inner (resp. outer) interface. For example, the codomain of  $F$  and domain of  $G$  is written as  $\langle 2, \{z, y\} \rangle$ .

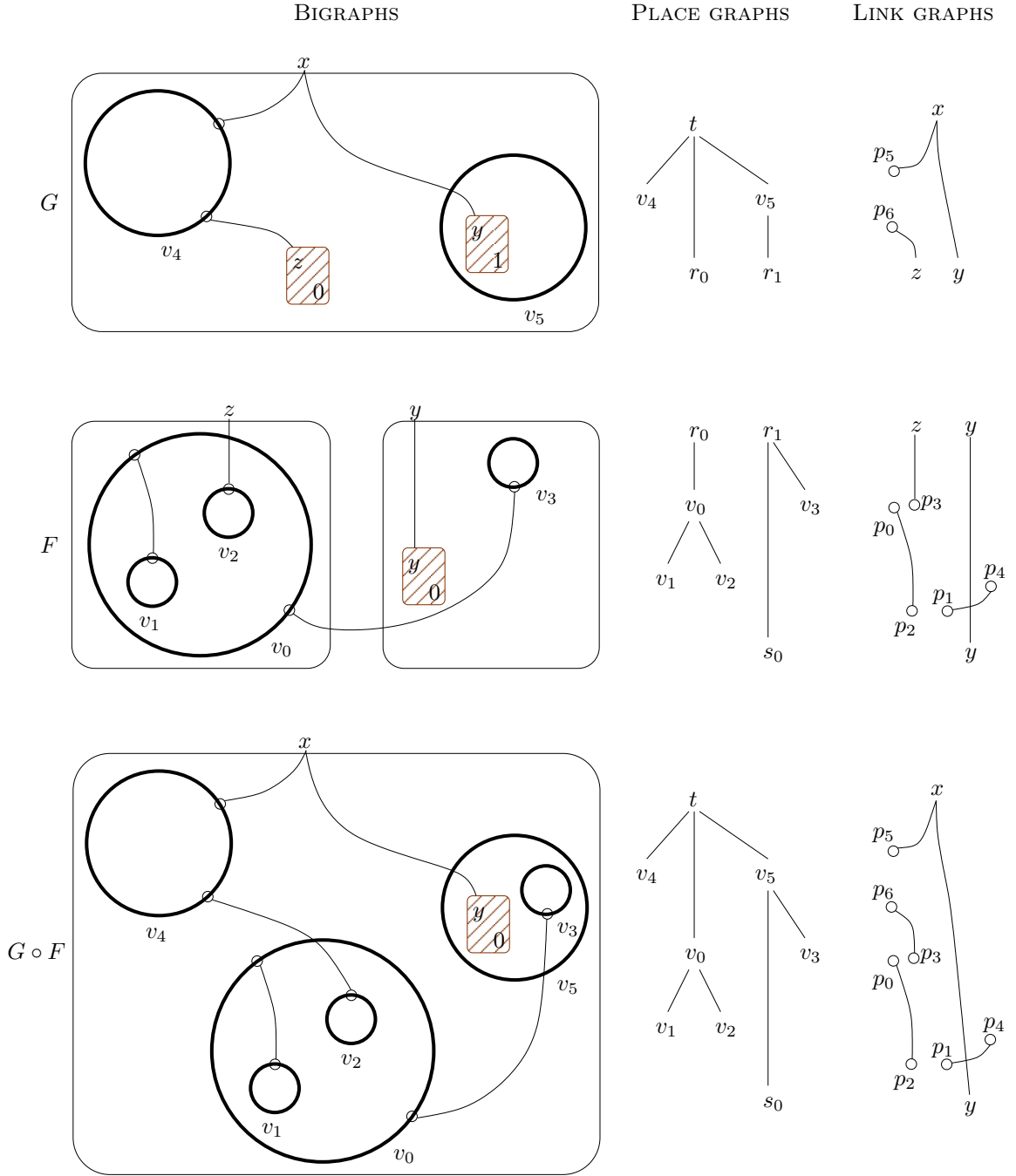


**Figure 2.4:** *Bigraphs, place graphs, link graphs, and composition*

Knowing the domain and codomain of a bigraph tells us little – the arrows themselves contain all the structure. We reproduce the formal definitions below [114]. However, similarly to cographs

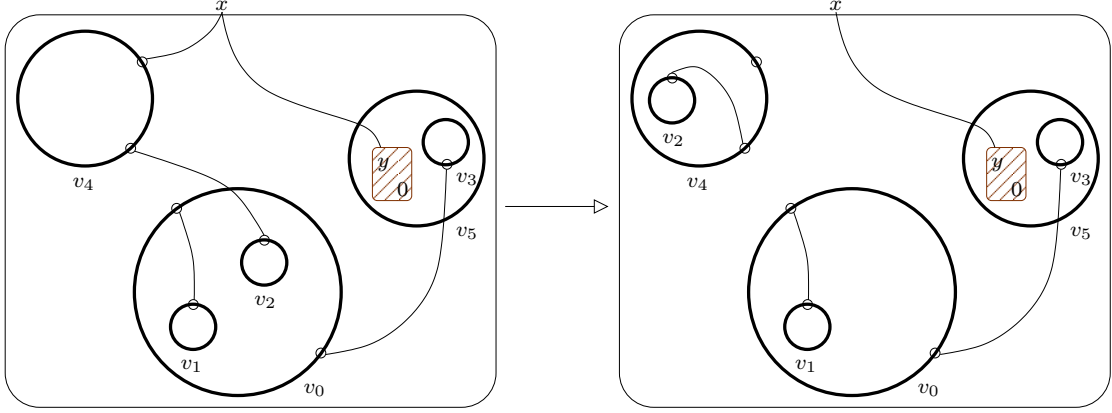


Figure 2.5: A bigraph reacts

of functions [88], the internal diagrams of bigraphs depicted in the figures are very informative themselves and we usually present them instead of formal terms.

Bigraphs are not just static structures; they are a framework for modelling mobile computation. An s-category of bigraphs can be equipped with a set of *reaction rules*. These reaction rules generate a reaction relation which describes how the bigraphs may reconfigure themselves. The s-category and its reaction rules together describe a bigraphical reactive system (Brs).<sup>8</sup> The reaction rules may be parametric, meaning that only part of the redex and reactum is specified in the rule and that an actual redex is formed only when the parameters are supplied. This is a familiar notion; consider the substitution generation rule  $(\lambda x.t) u \rightarrow_b t[x/u]$  of many explicit substitution calculi. The parameters here are the subterms  $t$  and  $u$ . Certain controls may prohibit reaction below them *e.g.* a guarded term of a process calculus. This is specified in the signature. Figure 2.5 depicts a reaction where the location of  $v_2$  has changed while its link with  $v_4$  is preserved and  $v_4$  has lost its connection with the name  $x$ .

Our presentation of dynamic signature below differs (*i.e.* it has an explicit *kind* function) but is equivalent to the standard definition. We chose this presentation as it better highlights the changes we consider in the following chapters.

**Definition 2.27** (dynamic signature). A dynamic signature  $\mathcal{K} \{\mathcal{K}, \text{arity}, \text{actv}, \text{kind}\}$  is composed of a set  $\mathcal{K}$  of controls and three maps:

$$\text{arity} : \mathcal{K} \rightarrow \mathbb{N} \qquad \text{actv} : \mathcal{K} \rightarrow \{\text{passive}, \text{active}\} \qquad \text{kind} : \mathcal{K} \rightarrow \{\emptyset, \mathcal{K}\}.$$

For each control  $K$ , arity provides a natural number  $\text{ar}(K)$ , an arity (the number of ports of the

<sup>8</sup>We are purposefully simplifying the situation here.

control). The *actv* function determines which controls allow reaction inside them (active controls) and which do not (passive controls). If  $\text{kind}(K) = \mathcal{K}$  then  $K$  may contain other controls or sites and is called non-atomic. Otherwise, it may not contain any control or site in a bigraph and is termed atomic.

Atomic controls cannot contain other controls and so, along with passive controls, do not allow reaction inside them; active controls do.

Finally, the operational semantics of a Brs is given by the reaction relation and a labelled transition system (LTS). This is where the general framework becomes quite powerful. There is a standard notion of label (as context) for Brss based on RPOs and hence canonical labelled transition systems. This avoids the need to manually (*ad hoc*) derive an LTS. The LTSs of the encodings of the calculi mentioned in the introduction correspond closely to the original calculi. We discuss labels in more detail in Chapter 5.

**Notation.** When  $\vec{A}$  denotes a pair of arrows  $A_0$  and  $A_1$ , we let  $i$  range over  $\{0, 1\}$ ,  $\bar{i} = 1 - i$ , and write  $A_i$  to denote one of the pair and  $A_{\bar{i}}$  the other.

### 2.3.1 Place graphs

Place graphs are defined over a dynamic signature  $\mathcal{K}$ .

**Definition 2.28** (place graph). A place graph  $A = (V, \text{ctrl}, \text{prnt}) : m \rightarrow n$  has an inner width  $m$  and an outer width  $n$ , both finite ordinals; a finite set  $V$  of nodes with a control map  $\text{ctrl} : V \rightarrow \mathcal{K}$ ; and a parent map  $\text{prnt} : m \uplus V \rightarrow V \uplus n$ .  $w >_A w'$ , or just  $w > w'$ , means that  $w = \text{prnt}^k(w')$  for some  $k > 0$ . We also write  $w >_A^1 w'$  if  $w = \text{prnt}(w')$  and let  $A(w)$  denote  $\text{prnt}(w)$ . The parent map is acyclic, i.e.  $\text{prnt}^k(v) \neq v$  for all  $k > 0$  and  $v \in V$ . An atom, i.e. a node with atomic control, may not be a parent.

The widths  $m$  and  $n$  index the *sites* and *roots* of  $A$  respectively. The sites and nodes – i.e. the domain of  $\text{prnt}$  – are called *places*. Nodes with atomic controls may not parent nodes or sites in a place graph. A place graph is *hard* if every root, and every node except an atom, has a child.

**Notation** (sites). We use  $s$ ,  $r$ , and  $t$  to denote sites and roots of place graphs.

**Definition 2.29** (s-category of place graphs). The s-category  $\mathcal{PLG}$  has finite ordinals as objects and place graphs as arrows. The support of a place graph is its node set. The composition  $A_1 A_0 : m_0 \rightarrow m_2$  of two place graphs  $A_i = (V_i, \text{ctrl}_i, \text{prnt}_i) : m_i \rightarrow m_{i+1}$  ( $i = 0, 1$ ) with disjoint supports is

$A_1 A_0 \stackrel{\text{def}}{=} (V, \text{ctrl}, \text{prnt})$  where  $V = V_0 \uplus V_1$ ,  $\text{ctrl} = \text{ctrl}_0 \uplus \text{ctrl}_1$ , and  $\text{prnt} = (\text{Id}_{V_0} \uplus \text{prnt}_1) \circ (\text{prnt}_0 \uplus \text{Id}_{V_1})$ . The identity place graph at  $m$  is  $\text{id}_m \stackrel{\text{def}}{=} (\emptyset, \emptyset_{\mathcal{K}}, \text{Id}_m) : m \rightarrow m$ .

The tensor product  $\otimes$  in  $\text{PLG}$  is defined as follows: On objects,  $m \otimes n \stackrel{\text{def}}{=} m + n$ . For the product  $A_0 \otimes A_1$  of two place graphs with disjoint support, the support of the product is the union of their node set; for the parent map, if  $A_0 : m_0 \rightarrow n_0$ , first offset the sites and roots of  $A_1$  by  $m_0$  and  $n_0$  respectively, then take the union of the two parent maps. The sub-s-category  $\text{PLG}_h$  consists of hard place graphs.

For an injective map  $\rho$  on nodes, the support translation  $\rho \bullet A$  is defined by systematic replacement of each node  $v$  by  $\rho(v)$ , preserving all structure.

**Definition 2.30** (barren, sibling, active, passive). A node or root is *barren* if it has no children. Two sites are *siblings* if they have the same parent. A site  $s$  of  $A$  is *active* if  $\text{ctrl}(v)$  is active whenever  $v > s$ ; otherwise  $s$  is *passive*. If  $s$  is active (resp. passive) in  $A$ , it will also be said that  $A$  is active (resp. passive) at  $s$ .

**Definition 2.31** (isomorphisms, epis, monos, inner-injective). An arrow  $\iota : m \rightarrow m$  in  $\text{PLG}$  is an isomorphism iff it has no nodes, and its parent map is a bijection. In  $\text{PLG}$ , a place graph is an *epi* iff no root is barren; it is *mono* iff no two sites are siblings. A place graph is *inner-injective* if no two sites are siblings.

### 2.3.2 Link graphs

Link graphs are defined over a dynamic signature  $\mathcal{K}$ .

**Definition 2.32** (link graph). A link graph  $A = (V, E, \text{ctrl}, \text{link}) : X \rightarrow Y$  has finite sets  $X$  of inner names,  $Y$  of (outer) names,  $V$  of nodes, and  $E$  of edges. It also has a function  $\text{ctrl} : V \rightarrow \mathcal{K}$  called the control map, and a function  $\text{link} : X \uplus P \rightarrow E \uplus Y$  called the link map, where  $P \stackrel{\text{def}}{=} \sum_{v \in V} \text{ar}(\text{ctrl}(v))$  is the set of ports of  $A$ .

The inner names  $X$  and ports  $P$  are called the points of  $A$ , and the edges  $E$  and outer names  $Y$  are called the links of  $A$ .

**Definition 2.33** (s-category of link graphs). The s-category  $\text{LIG}$  has name sets as objects and link graphs as arrows. The composition  $A_1 \circ A_0 : X_0 \rightarrow X_2$  of two link graphs  $A_i = (V_i, E_i, \text{ctrl}_i, \text{link}_i) : X_i \rightarrow X_{i+1}$  ( $i = 0, 1$ ) is defined when their support, their node sets and edge sets, are disjoint; then  $A_1 \circ A_0 \stackrel{\text{def}}{=} (V, E, \text{ctrl}, \text{link})$  where  $V = V_0 \uplus V_1$ ,  $\text{ctrl} = \text{ctrl}_0 \uplus \text{ctrl}_1$ ,  $E = E_0 \uplus E_1$  and  $\text{link} = (\text{Id}_{E_0} \uplus \text{link}_1) \circ (\text{link}_0 \uplus \text{Id}_{P_1})$ . The identity link graph at  $X$  is  $\text{id}_X \stackrel{\text{def}}{=} (\emptyset, \emptyset, \emptyset_{\mathcal{K}}, \text{Id}_X) : X \rightarrow X$ .

The tensor product of two objects  $X$  and  $Y$  in  $\mathcal{LIG}$  is defined as the union of sets required to be disjoint. The tensor product  $A_0 \otimes A_1 : X_0 \otimes X_1 \rightarrow Y_0 \otimes Y_1$  of two link graphs  $A_i : X_i \rightarrow Y_i, i \in \{0, 1\}$  is defined when the interface products are defined and when  $A_0$  and  $A_1$  have disjoint support. The product is then formed by taking the union of their link maps.

**Definition 2.34** (idle, open, closed, peer, lean). A link is idle if it has no preimage under the link map. An (outer) name is an open link, an edge is a closed link. A point (i.e. an inner name or port) is open if its link is open, otherwise closed. Two distinct points are peers if they are in the same link. A link graph is lean if it has no idle edges.

### 2.3.3 Bigraphs

**Notation.** A natural number  $m$  is often interpreted as a finite ordinal  $m = \{0, 1, \dots, m-1\}$ .  $i$  is often used to range over  $m$ ; when  $m = 2$ ,  $\bar{i}$  is used for the complement  $1 - i$  of  $i$ .  $\vec{x}$  denotes a finite sequence  $\{x_i \mid i \in m\}$ ; when  $m = 2$  this is an ordered pair.

**Definition 2.35** (concrete pure bigraph). A (concrete) pure bigraph over the signature  $\mathcal{K}$  takes the form  $G = (V, E, ctrl, G^P, G^L) : I \rightarrow J$  where  $I = \langle m, X \rangle$  and  $J = \langle n, Y \rangle$  are its inner and outer faces, each combining a width (a finite ordinal) with a finite set of global names drawn from a denumerable set  $\mathcal{X}^9$ . Its first two components  $V$  and  $E$  are finite sets of nodes and edges respectively. The third component  $ctrl : V \rightarrow \mathcal{K}$ , a control map, assigns a control to each node. The remaining two are:

$$G^P = (V, ctrl, prnt) : m \rightarrow n \quad \text{a place graph}$$

$$G^L = (V, E, ctrl, link) : X \rightarrow Y \quad \text{a link graph.}$$

$G$  is called the combination of its constituents  $G^P$  and  $G^L$ ; it is written as  $G = \langle G^P, G^L \rangle$ .

**Definition 2.36** (s-category of pure concrete bigraphs). The s-category  $\mathcal{BIG}(\mathcal{K})$  of pure concrete bigraphs over a signature  $\mathcal{K}$  has interfaces  $I = \langle m, X \rangle$  as objects, with origin  $\epsilon = \langle 0, \emptyset \rangle$ , and bigraphs  $G : I \rightarrow J$  as arrows.  $I$  is called the inner face of  $G$ , and  $J$  the outer face. The support set,  $|G|$ , of  $G$  is the disjoint union of the sets of nodes and edges of  $G$ . If  $F : J \rightarrow K$  is another bigraph with  $|F| \cap |G| = \emptyset$ , then their composition is defined directly in terms of the compositions of the constituents as follows:

$$FG \stackrel{\text{def}}{=} \langle F^P G^P, F^L G^L \rangle : I \rightarrow K.$$

---

<sup>9</sup>This denumerable set  $\mathcal{X}$  of global names is presupposed in the theory.

The identities are  $\langle \text{id}_m, \text{id}_X \rangle : I \rightarrow I$ , where  $I = \langle m, X \rangle$ . The sub-s-category  $\mathcal{BIG}_h$  consists of hard bigraphs, those with place graphs in  $\mathcal{PLG}_h$ .

We leave the definition of the tensor product separate as we will reference it directly.

**Definition 2.37** (tensor product). The tensor product of two bigraph interfaces is defined by  $\langle m, X \rangle \otimes \langle n, Y \rangle \stackrel{\text{def}}{=} \langle m+n, X \uplus Y \rangle$  when  $X$  and  $Y$  are disjoint. The tensor product of two bigraphs  $G_i : I_i \rightarrow J_i (i = 0, 1)$  is defined by

$$G_0 \otimes G_1 \stackrel{\text{def}}{=} \langle G_0^P \otimes G_1^P, G_0^L \otimes G_1^L \rangle : I_0 \otimes I_1 \rightarrow J_0 \otimes J_1$$

when the interfaces exist and the node and edge sets are disjoint. This combination is well-formed, since its constituents share the same node set.

**Notation.** When considering a span  $\vec{A} : h \rightarrow \vec{m}$  of bigraphs, a convention is adopted for naming their nodes, edges, roots, and outer names. The node set of  $A_i (i = 0, 1)$  is denoted by  $V_i$ , and  $V_0 \cap V_1$  is denoted by  $V_2$ .  $w_2, w'_2, \dots$  will be used to range over  $h \uplus V_2$ . Similarly, the edge sets are denoted by  $E_i$ .  $r_i, r'_i, \dots$  will be used to range over the roots  $m_i, i \in 2$ ,  $x_i, x'_i, \dots$  will be used to range over the names  $X_i, i \in 2$ , and  $v_i, v'_i, \dots$  will be used to range over  $V_i, i \in 3$  where  $V_3 \cap (V_1 \cup V_2) = \emptyset$ . We use similar notation for cospans of bigraphs.

**Notation.** When dealing with many place (resp. link) graphs  $A, B, \dots$ , instead of indexing their parent maps as  $\text{prnt}_A, \text{prnt}_B$  (resp.  $\text{link}_A, \text{link}_B$ ) etc. it will be more convenient to abuse notation and denote these maps by  $A$  as in Definition 2.28 e.g.  $A(v) = \text{prnt}_A(v)$ ,  $A(x) = \text{link}_A(x)$ .

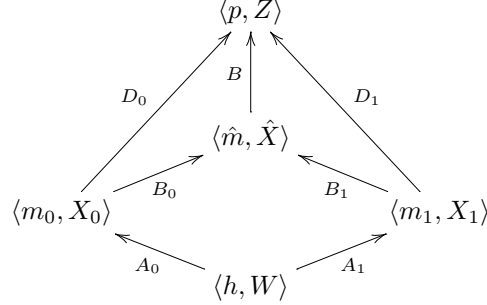
**Remark** (bigraphs are wide).  $\mathcal{BIG}(\mathcal{K})$  and  $\mathcal{BIG}_h(\mathcal{K})$  are wide s-categories. The interface  $I = \langle n, X \rangle$  has  $\text{width}(I) = n$ , and for  $G : \langle m, X \rangle \rightarrow \langle n, Y \rangle$  the width map  $\text{width}(G)$  sends each site  $i \in m$  to the unique root  $j \in n$  such that  $i <_G j$ .

We reproduce the RPO construction for pure bigraphs below as we refer to it later.

**Construction 2.38** (RPOs in pure bigraphs [74]).

An RPO  $\vec{B} : \langle \vec{m}, \vec{X} \rangle \rightarrow \langle \hat{m}, \hat{X} \rangle, B : \langle \hat{m}, \hat{X} \rangle \rightarrow \langle p, Z \rangle$ , for a pair  $\vec{A} : \langle h, W \rangle \rightarrow \langle \vec{m}, \vec{X} \rangle$  of bigraphs relative to a bound  $\vec{D} : \langle \vec{m}, \vec{X} \rangle \rightarrow \langle p, Z \rangle$ , is built in four stages.





**nodes and edges:** If  $V_i$  are the nodes of  $A_i$  ( $i = 0, 1$ ) then the nodes of  $D_i$  are  $V_i - V_2 \uplus V_3$  for some  $V_3$ . Define the nodes of  $B_i$  and  $B$  to be  $V_i - V_2$  ( $i = 0, 1$ ) and  $V_3$  respectively. Edges  $E_i$  are treated analogously and ports  $P_i$  inherit the analogous treatment from nodes.

**interface:** Construct the shared codomain  $\langle \hat{m}, \hat{X} \rangle$  of  $\vec{B}$  as follows.

For  $\hat{m}$ , define the roots in each  $m_i$  that must be mapped into  $\hat{m}$ :

$$m'_i \stackrel{\text{def}}{=} \{r \in m_i \mid D_i(r) \in V_3 \uplus p\}.$$

Define  $\cong_P$  as the smallest equivalence on the disjoint sum  $m'_0 + m'_1$  for which  $(0, r_0) \cong_P (1, r_1)$  whenever  $A_0(w) = r_0$  and  $A_1(w) = r_1$  for some  $w \in h \uplus V_2$ . Define  $\hat{m}$  up to isomorphism:

$$\hat{m} \stackrel{\text{def}}{=} (m'_0 + m'_1) / \cong_P.$$

For each  $r \in m'_i$  we denote the  $\cong_P$ -equivalence class of  $(i, r)$  by  $\widehat{i, r}$ .

For  $\hat{X}$ , define the names in each  $X_i$  that must be mapped into  $\hat{X}$ :

$$X'_i \stackrel{\text{def}}{=} \{x \in X_i \mid D_i(x) \in P_3 \uplus Z\}.$$

Define  $\cong_L$  as the smallest equivalence on the disjoint sum  $X'_0 + X'_1$  for which  $(0, x_0) \cong_L (1, x_1)$  whenever  $A_0(p) = x_0$  and  $A_1(p) = x_1$  for some point  $p \in W \uplus P_2$ . Define  $\hat{X}$  up to isomorphism:

$$\hat{X} \stackrel{\text{def}}{=} (X'_0 + X'_1) / \cong_L.$$

For each  $x \in X'_i$  we denote the  $\cong_L$ -equivalence class of  $(i, x)$  by  $\widehat{i, x}$ .

For the parents and links, we present the construction for  $B_0$ .  $B_1$  is constructed similarly.

**parents:** Define  $B_0$  to simulate  $D_0$  as far as possible

$$\begin{aligned} \text{For } r \in m_0 : \quad B_0(r) &\stackrel{\text{def}}{=} \begin{cases} \widehat{0, r} & \text{if } r \in m'_0 \\ D_0(r) & \text{if } r \notin m'_0 \end{cases} \\ \text{For } v \in V_1 - V_2 : \quad B_0(v) &\stackrel{\text{def}}{=} \begin{cases} \widehat{1, r} & \text{if } A_1(v) = r \in m_1 \\ D_0(v) & \text{if } A_1(v) \notin m_1 \end{cases} \end{aligned}$$

Define  $B$  to simulate both  $D_0$  and  $D_1$ :

$$\begin{aligned} \text{For } \hat{r} \in \hat{m} : \quad B(\hat{r}) &\stackrel{\text{def}}{=} D_i(r) \text{ where } \widehat{i, r} = \hat{r} \\ \text{For } v \in V_3 : \quad B(v) &\stackrel{\text{def}}{=} D_i(v). \end{aligned}$$

**links:** Define  $B_0$  to simulate  $D_0$  as far as possible:

$$\begin{aligned} \text{For } x \in X_0 : \quad B_0(x) &\stackrel{\text{def}}{=} \begin{cases} \widehat{0, x} & \text{if } x \in X'_0 \\ D_0(x) & \text{if } x \notin X'_0 \end{cases} \\ \text{For } p \in P_1 - P_2 : \quad B_0(p) &\stackrel{\text{def}}{=} \begin{cases} \widehat{1, x} & \text{if } A_1(p) = x \in X_1 \\ D_0(p) & \text{if } A_1(p) \notin X_1. \end{cases} \end{aligned}$$

Define  $B$  to simulate both  $D_0$  and  $D_1$ :

$$\begin{aligned} \text{For } \hat{x} \in \hat{X} : \quad B(\hat{x}) &\stackrel{\text{def}}{=} D_i(x) \text{ where } x \in X_i \text{ and } \widehat{i, x} = \hat{x} \\ \text{For } p \in P_3 : \quad B(p) &\stackrel{\text{def}}{=} D_i(p). \end{aligned}$$

□

Consistency conditions for bigraphs are given in Appendix A.1.

**Definition 2.39** (discrete). *A bigraph is discrete if it has no edges and a bijective link map. The link map is then defined as  $\text{link} : X \uplus P \rightarrow Y$  and  $|X| + |P| = |Y|$ . This means that every point is open, no two points are peers, and no name is idle.*

**Definition 2.40** (wiring, substitution, closure). *A bigraph with zero width is called a wiring. Arbitrary wirings are denoted by  $\omega$ . All wirings can be built from the composition and tensor product of two basic types of wiring: i) closures  $/x : x \rightarrow \epsilon$  which hide a name from the outer interface; and ii) functions  $\sigma : X \rightarrow Y$  called substitutions. The empty substitution  $x : \epsilon \rightarrow x$  (substitutions need not be surjective) adds a name to the outer interface.*

**Notation.** *We often omit  $\dots \otimes \text{id}_I$  in compositions, when there is no ambiguity; for example, given  $G : \langle m, \emptyset \rangle \rightarrow \langle n, X \rangle$  and  $\text{merge} : \langle m, \emptyset \rangle \rightarrow \langle 1, \emptyset \rangle$ , we write  $\text{merge} \circ G$  for  $(\text{merge} \otimes \text{id}_X) \circ G$ .*

*Given a wiring  $\omega : \langle 0, Y \rangle \rightarrow \langle 0, Z \rangle$  we may restrict its link map to any subset  $X \subseteq Y$ , yielding the restricted wiring  $\omega \upharpoonright X : \langle 0, X \rangle \rightarrow \langle 0, Z \rangle$ . Then, if the outer face of  $G$  is  $\langle m, X \rangle$  we may simply write  $\omega G$  for  $(\omega \upharpoonright X \otimes \text{id}_m) \circ G$ .*

**Definition 2.41** (prime). *A bigraph with width one is called a prime bigraph.*

**Definition 2.42** (ion, atom, molecule). *The discrete ion  $K_{\vec{x}} : 1 \rightarrow \langle \vec{x} \rangle$  of a non-atomic control  $K$  with arity  $k$  is a prime bigraph with a single  $K$ -node, each port of which is linked to one of the  $k$  names in  $\vec{x}$  such that the linking is bijective. The discrete atom  $K_{\vec{x}} : \epsilon \rightarrow \langle \vec{x} \rangle$  is defined similarly but has no site. A discrete molecule is a composition  $(K_{\vec{x}} \otimes \text{id}_Y) \circ P$  of a discrete ion with a discrete prime  $P$  with outer names  $Y$ . An arbitrary ion, atom, or molecule is a prime bigraph constructed from a discrete ion, atom, or molecule by composing it with  $\omega \otimes \text{id}_1$  for some wiring  $\omega$ .*

**Notation** (term language, holes). *We use a term language to describe bigraphs throughout the dissertation which has previously been axiomatised for pure bigraphs [108]. The language consists of atoms, molecules, wirings, and holes, joined together by composition and tensor. The holes specify where sites are placed in a bigraph and are denoted by either  $\square$  or  $-$ . If a bigraph has more than one site then the holes are subscripted according to their order in the inner interface.*

For example, see the term in Figure 3.2 on page 49 which includes closures, substitutions, atoms, molecules, and holes.

We make use of a refinement of pure bigraphs by Milner called local bigraphs [106]. This refinement allows name binding as in the  $\lambda$ -calculus by partitioning the ports of controls into *free* ports and *binding* ports and enforcing conditions on links involving binding ports. The following definitions are taken from Milner’s work with minor modifications for brevity.

**Definition 2.43** (local interface). *A local interface takes the form  $I = \langle m, \text{loc}_I, X \rangle$ , where the width  $m$  and name set  $X$  are as in pure bigraphs, and  $\text{loc}_I \subseteq m \times X$  is called the locality of  $I$  and relates each  $x \in X$  with at least one  $i \in m$ . If  $(i, x) \in \text{loc}_I$  then  $i$  is a place of  $x$  (in  $I$ ). The interface  $I^\mathbf{u} = \langle m, X \rangle$  is called the pure interface underlying  $I$ .*

**Definition 2.44** (local bigraph). *A local bigraph  $G : I \rightarrow J$ , where  $I$  and  $J$  are local interfaces, consists of an underlying pure bigraph  $G^\mathbf{u} : I^\mathbf{u} \rightarrow J^\mathbf{u}$  satisfying the binding and scoping locality conditions below. The scoping condition employs the relation  $\text{loc}_G$ , the smallest relation on pure bigraphs which assigns places to points and links as follows:*

- points**    if  $(i, x) \in \text{loc}_I$  then  $(i, x) \in \text{loc}_G$   
               if  $p$  is a binding port of node  $v$  then  $(v, p) \in \text{loc}_G$   
               if  $p$  is a free port of node  $v$  then  $(\text{prnt}_G(v), p) \in \text{loc}_G$ .
- links**     if  $(j, y) \in \text{loc}_J$  then  $(j, y) \in \text{loc}_G$   
               if an edge  $e$  contains a binding port of  $v$  then  $(v, e) \in \text{loc}_G$ .

In the following locality conditions,  $q$  and  $l$  range over points and links respectively and  $w$  and  $w'$  range over places.

**BINDING:** A link has at most one binding port; an open link has none.

**SCOPING:** If  $\text{link}_G(q) = l$  then whenever  $(w, q) \in \text{loc}_G$  then there exists  $w'$  such that  $w \leq_G w'$  and  $(w', l) \in \text{loc}_G$ .

Local bigraphs form wide s-categories.

### 2.3.4 Related work

Aspects of bigraphs have been inspired by the interaction nets of Lafont [86]. Leifer and Milner's bigraphical model of *arithmetic nets* demonstrates part of this connection [92]. Both paradigms have the notion of *interface preservation*, where the (outer) interface is preserved through reduction. In the other direction, Fernández, Mackie, and Sinot [56] present an encoding of the  $\rho$ -calculus in *bigraphical nets*. Bigraphical nets add the notion of locality (via nesting *i.e.* place graph) to interaction nets. The encoding takes advantage of wide reaction.

Wide reaction is present in other frameworks. In particular, models of the  $\lambda$ -calculus which precede Milner's model (see Chapter 11) have a notion of wide/non-local substitution *e.g.* Ariola and Felleisen's call-by-need lambda calculus [5] and Ariola and Klop's cyclic  $\lambda$ -calculus [6]. Bertolissi et al.'s  $\rho_g$ -calculus, inspired by Ariola and Klop's work, extends the  $\rho$ -calculus to handle graph-like structures with cycles and sharing and is a natural extension of the cyclic  $\lambda$ -calculus [11, 12].

The  $\rho_g$ -calculus allows non-local or *external* substitution as do bigraphs and is confluent under linearity conditions [11]. In Part II of the dissertation, we (indirectly) study non-local substitution for Milner's model of the  $\lambda$ -calculus and present a proof of confluence. However, general proofs of confluence for bigraphs remain an open problem although Milner has identified conditions for local confluence and suggested the study of finite developments of reaction [111].

## 2.4 Term rewriting systems

Term rewriting systems have been applied in many areas: the Knuth-Bendix algorithm [85] for solving, in certain cases, the word problem of equational logic; functional programming languages; automatic theorem provers/assistants. There are many good introductions to the subject [71, 49, 83, 142] but we only require some basic concepts. We follow the definitions and notation of

Klop [83] here and favour the term *reduction* over *rewrite* however we admit that this terminology is not always intuitive as ‘reductions’ do not necessarily reduce terms.

**Definition 2.45** (abstract reduction system). *An abstract reduction system consists of a set  $A$  and a set of binary relations  $\{\rightarrow_{\alpha_1}, \dots, \rightarrow_{\alpha_2}\}$  on  $A$  called reduction rules. For  $a, b \in A$ , if  $(a, b) \in \rightarrow_{\alpha_i}$  then we write  $a \rightarrow_{\alpha_i} b$ . We call  $a$  and  $b$  the *redex* and *reactum* of the rule respectively.*

In order to ease the presentation, we assume a set  $A$  with elements  $a, b, \dots$  and an arbitrary reduction relation  $\rightarrow$  on  $A$  in the following.

**Notation** (closures and unions of relations, reduct). *The reflexive closure of  $\rightarrow$  is written as  $\rightarrow^=$ . The transitive closure of  $\rightarrow$  is written as  $\rightarrow^+$ . The transitive, reflexive closure of  $\rightarrow$  is written as  $\rightarrow^*$  or  $\rightarrow^n$ . We write  $\rightarrow^n$  to denote  $n$  consecutive reductions of  $\rightarrow$ . The union of two relations  $\rightarrow_\alpha$  and  $\rightarrow_\beta$  on a set  $A$  is denoted by  $\rightarrow_{\alpha\beta}$ . If  $a \rightarrow b$  then we say that  $b$  is a *reduct* of  $a$ .*

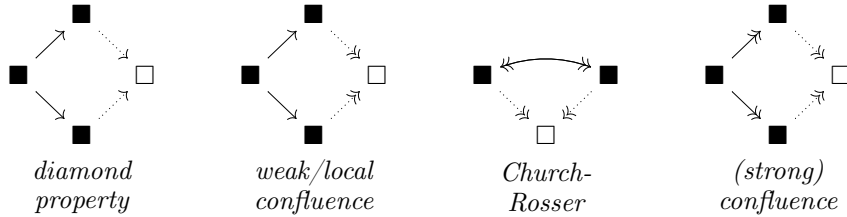
We use Rosen’s diagrammatic notation for rewriting proofs [134]. Rose gives a formal definition [132] but we take a somewhat informal approach here.

**Notation** (diagrammatic propositions). *A diagram is a directed graph consisting of solid and unfilled nodes, or boxes, and solid and dotted labelled arrows between nodes. The boxes represent arbitrary terms of a calculus and the arrows represent relations between terms. The solid boxes and arrows represent known or given statements, the antecedents of a hypothesis. The unfilled boxes and dotted arrows represent the consequents of a hypothesis.*

For example, the diagram of the diamond property in the next definition represents the proposition “if  $a \rightarrow b_1$  and  $a \rightarrow b_2$  then there exists  $c$  such that  $b_1 \rightarrow c$  and  $b_2 \rightarrow c$ .”

The study of confluence is common in rewriting theory. Informally, we say that  $a$  is confluent if given two distinct reducts  $b_1$  and  $b_2$  of  $a$ ,  $b_1$  and  $b_2$  have a common reduct  $c$  which ‘closes’ the reduction sequence. There are different degrees of confluence. The following diagrams depict the terminology we use.

**Definition 2.46** (confluence).



The double-headed arrow in the Church-Rosser diagram denotes the equivalence closure of  $\rightarrow$ <sup>10</sup>.

Weak confluence states that when a term  $a$  can reduce to two other terms  $b_1$  and  $b_2$  then there exists a common reduct,  $c$ , of the latter pair where  $b_1 \twoheadrightarrow c$  and  $b_2 \twoheadrightarrow c$ . The other properties are defined similarly. Milner [107] calls the diamond property *one-step confluence* and weak confluence has also been called *weak Church-Rosser* or *local confluence*. Any relation is confluent if and only if it has the Church-Rosser property [37] and we will use these terms interchangeably.

There are further distinctions; *closed confluence* refers to confluence on terms without metavariables, *open confluence* also considers terms with metavariables. As we will not discuss metavariables and all confluence properties studied here are closed, we will omit the qualifier.

**Definition 2.47** (normalisation). *The normal forms of  $A$  are those elements  $a \in A$  such that no  $b \in A$  exists where  $a \rightarrow b$ . We say that an element  $b$  has a normal form if  $b \twoheadrightarrow a$  where  $a$  is a normal form.*

*The relation  $\rightarrow$  on  $A$  is weakly normalising if every element of  $A$  has a normal form. The relation  $\rightarrow$  on  $A$  is strongly normalising (or terminating or noetherian) if every reduction sequence  $a \rightarrow b \rightarrow \dots$  is finite. The relation  $\rightarrow$  has unique normal forms if for all elements  $a$ , all normal forms of  $a$  are equal. The same terminology extends to abstract rewrite systems where the union of the reduction relations satisfies these properties.*

**Notation.** *When a reduction relation  $\rightarrow$  has unique normal forms, we write  $\downarrow(a)$  for the  $\rightarrow$ -normal form of a term  $a$  and we write  $\twoheadrightarrow$  to denote reduction to normal form.*

**Notation.** *We write  $\rightarrow SN$  when  $\rightarrow$  is strongly-normalising,  $\rightarrow CR$  when it has the Church-Rosser property,  $\rightarrow LC$  when it has weak confluence,  $\rightarrow \Diamond$  when it has the diamond property, and  $\rightarrow UN$  when it has unique normal forms.*

**Notation** ( $SN_R, WN_R$ ). *When discussing a rewrite system with reduction relation  $R$ ,  $SN_R$  denotes the set of strongly normalising terms and  $WN_R$  denotes the set of weakly normalising terms.*

**Definition 2.48** (reduction modulo equivalence). *Let  $\sim$  and  $\rightarrow$  respectively be a congruence and a reduction relation on a set  $A$  of terms. The relation  $\rightarrow/\sim$  of reduction  $\rightarrow$  modulo  $\sim$  is defined by*

$$a \rightarrow/\sim a' \quad \text{iff there exist } b, b' \text{ s.t. } a \sim b \rightarrow b' \sim a'.$$

<sup>10</sup>“This is not always obtainable by constructing the reflexive, transitive and then symmetric closures in some sequence, of course, even if the [double-headed arrow notation] might be interpreted to indicate just that.” [132]

## 2.5 The $\lambda$ -calculus

The  $\lambda$ -calculus [36] is one of the best-known term rewriting systems in computer science. It is a compact formalisation of the computable functions, having a simple grammar and only one reduction rule,  $\beta$ -reduction. We cannot hope to properly summarise the theory of the  $\lambda$ -calculus or its contributions (which include functional programming languages and denotational semantics) here so we refer the reader to the standard text [8].

The set  $\Lambda$  of  $\lambda$ -terms is inductively defined by

$$t ::= x \mid \lambda x.t \mid t t$$

where  $x$  ranges over a denumerable set  $\mathcal{X}$  of variables, the constructor  $\lambda x.t$  is called an *abstraction*, and the final constructor an *application*. We use  $t, u$ , and  $v$  to denote terms,  $x, y$ , and  $z$  to denote variables. We use the same notation for explicit substitution calculi.

Variables in terms may either be *free* or *bound*. A variable  $x$  is bound if it lies inside an abstraction  $\lambda x.t$ . We denote the free variables of a term  $t$  by  $\text{FV}(t)$  and the bound variables by  $\text{BV}(t)$ . They are defined inductively.

**Definition** (free and bound variables).

$$\begin{aligned} \text{FV}(x) &= \{x\} & \text{BV}(x) &= \emptyset \\ \text{FV}(\lambda x.t) &= \text{FV}(t) \setminus \{x\} & \text{BV}(\lambda x.t) &= \text{BV}(t) \cup \{x\} \\ \text{FV}(t u) &= \text{FV}(t) \cup \text{FV}(u) & \text{BV}(t u) &= \text{BV}(t) \cup \text{BV}(u) \end{aligned}$$

**Definition 2.49** (fresh variable). *A variable  $x$  is said to be fresh for  $t$  if  $x \notin \text{FV}(t) \cup \text{BV}(t)$ .*

**Convention** (bound variables). *In all named calculi of explicit substitutions the bound names of a term are chosen to be distinct and different from the free names of the term.*

Throughout the dissertation, we implicitly assume the following convention, following Barendregt [8], and avoid the issues of variable capture and variable clash by considering terms up to  $\alpha$ -equivalence.

**Definition 2.50** ( $\alpha$ -equivalence  $\equiv$ ). *Two terms are said to be  $\alpha$ -equivalent if they are identical up to renaming of bound variables.*

**Definition 2.51** (substitution). *The substitution  $t\{x/u\}$  of a term  $u$  for all free occurrences of  $x$  in  $t$  is defined by:*

$$\begin{aligned} x\{x/u\} &= u & y\{x/u\} &= y & \text{where } y \neq x \\ (t_1 t_2)\{x/u\} &= t_1\{x/u\} t_2\{x/u\} & (\lambda y.t)\{x/u\} &= \lambda y.t\{x/u\} & \text{where } y \neq x \end{aligned}$$

The  $\lambda$ -calculus has one main reduction rule,  $\beta$ -reduction (we do not consider  $\eta$ -reduction here). We define this rule, and all rules of explicit substitution calculi in the dissertation, to be closed under contexts modulo  $\equiv$ .

**Definition** ( $\beta$ -reduction).  *$\beta$ -reduction is defined by the rule  $(\lambda x.t) u \rightarrow_{\beta} t\{x/u\}$ .*

$\beta$ -reduction describes the application of a function  $\lambda x.t$  to its argument  $u$ , where the result of the rule (the reactum) is that all free occurrences of  $x$  in  $t$  are replaced by  $u$  in a manner which avoids variable capture. We take  $\beta$ -reduction as the reduction relation of the  $\lambda$ -calculus.

**Theorem 2.52** ( $\rightarrow_{\beta}$  is confluent).

*Proof.* First proved by Church and Rosser [37]. □

We can extend the grammar of  $\Lambda$  with a *hole* constructor. A  $\lambda$ -context is defined as a  $\lambda$ -term with a single hole *e.g.*  $\square$  or  $\lambda x.\square$ . We let  $C[-]$  and  $D[-]$  denote contexts. Given a context  $C[-]$ , the term  $C[t]$  is obtained by filling in the hole of  $C$  with a term  $t$  *such that no free variables of  $t$  are bound by the context*.

**Definition 2.53** ((strict) subterm). *If  $t = C[u]$  then we call  $u$  a subterm of  $t$  and write  $u \subseteq t$ . If  $C \neq \square$  then  $u$  is a strict subterm in which case we write  $u \subset t$ .*

## 2.6 Explicit substitutions

The  $\lambda$ -calculus is the mathematical foundation behind most functional programming languages. However, while the  $\lambda$ -calculus is a purely mathematical system, programming languages are meant for practical purposes and as such are concerned about notions of complexity and efficiency. In particular, one cannot reason about the meta-operation of  $\lambda$ -calculus substitution in terms of time or space constraints. In functional programming languages, the applications of substitutions are often delayed and intertwined suggesting that a finer formalism than the  $\lambda$ -calculus is needed to reason about functional programs.

Rose [132] notes that solutions to this problem were proposed by Curry and Feys [43] and later by Curien [42]. This latter work led to the development of *explicit substitution (ES) calculi*, beginning with the  $\lambda\sigma$  calculus of Abadi et al. [1].

The study of *explicit substitutions* was initially concerned with breaking  $\beta$ -reduction down into simpler reduction steps, providing a formal bridge between the  $\lambda$ -calculus and functional



programming languages [ibid.]. Explicit substitutions have also been applied to bridge the gap between theorem provers and the calculi (with a substitution meta-operation) they are based on.

It can be shown that explicit substitution is related to the logical cut rule [51]. On the basis of this observation, close relations have been established between normalisation of some ES calculi and cut elimination in Linear Logic's proof nets by translating typable terms from the former into nets of the latter [52, 80, 78]. Simulations along these translations can then demonstrate, by reflection, normalisation properties of the calculi.

In order to present some of the terminology related with explicit substitutions, we take Kesner's approach of assuming an ES calculus  $\lambda_z$  with reduction relation  $\rightarrow_{\lambda_z}$  and a mapping  $\mathcal{E}$  from  $\lambda$ -terms to  $\lambda_z$ -terms. We will only consider named ES calculi. Syntactically, many of these simply extend the  $\lambda$ -calculus grammar with the constructor  $t[x/u]$  and the mapping  $\mathcal{E}$  is the identity. The notation  $[x/u]$  represents an explicit substitution. It is a *term construction* not be confused with the substitution meta-operation of the  $\lambda$ -calculus which we denote with curly brackets.

**Terminology** (pure terms). *The pure terms of an explicit substitution calculus  $\lambda_z$  are the images of  $\lambda$ -terms under  $\mathcal{E}$  i.e. they have no explicit substitutions.*

The sets of free and bound variables of an ES term are defined as usual, the additions being:

$$\text{FV}(t[x/u]) = (\text{FV}(t) \setminus \{x\}) \cup \text{FV}(u) \quad \text{BV}(t[x/u]) = \text{BV}(t) \cup \{x\} \cup \text{BV}(u).$$

The sets of reaction rules of the ES calculi we consider are the main differentiating factor between the calculi. Throughout this dissertation, we consider the reduction relation of an ES calculus to be the union of its reduction rules closed under contexts modulo  $\equiv$ . The exceptions to this rule are the  $\lambda_{\text{lr}}$  and  $\lambda_{\text{es}}$  calculi (see Chapter 7) which reduce modulo an equivalence relation which subsumes  $\alpha$ -equivalence.

The idea behind ES calculi is that  $\beta$ -reduction is broken down into simpler steps to allow some notion of operational analysis. Therefore, we would like the union of these simpler reductions to behave just like  $\beta$ -reduction *i.e.* if a term is confluent or strongly normalising in the  $\lambda$ -calculus then its  $\mathcal{E}$ -image should have the same properties in the ES calculus. However, the cost incurred by splitting the reduction relation is that the simpler reductions can interleave in ways which lose these properties.

The canonical counterexample of an expected normalisation property is Mellès' example that a simply typed  $\lambda$ -term (hence strongly normalising for  $\rightarrow_{\beta}$ ) was not strongly normalising in  $\lambda\sigma$  [98]. This unexpected result began the search for explicit substitution calculi which truly extend the  $\lambda$ -calculus operationally as well as syntactically.

**Definition 2.54** (Desirable properties of ES calculi). *Over the years, much research has been done trying to find an ES calculus with the following desirable properties [78]:*

- *Closed confluence: The reduction relation is confluent on terms without metavariables.*
- *Open confluence: The reduction relation is confluent on terms with metavariables.*
- *Preservation of strong normalisation of  $\beta$ -reduction (PSN): If  $t \in \text{SN}_\beta$  then  $\mathcal{E}(t) \in \text{SN}_{\lambda_z}$ .*
- *Strong normalisation of typable terms: If  $t$  in  $\lambda_z$  is typable for some type system then  $t$  is strongly normalising.*
- *Simulation of  $\beta$ -reduction: If  $t \rightarrow_\beta u$  then  $\mathcal{E}(t) \rightarrow_{\lambda_z} \mathcal{E}(u)$ .*
- *Full composition of substitutions (FCS):  $t[x/u] \rightarrow_{\lambda_z} t\{x/u\}$  where  $t\{x/u\}$  is defined as expected (up to  $\alpha$ -equivalence and avoiding variable capture).*

We will support our claim that  $\Lambda_{\text{sub}}$  (and hence  $\Lambda_{\text{BIG}}$ ) is operationally close to the  $\lambda$ -calculus by proving that it satisfies all of these properties. The proof of open confluence is not presented in the dissertation but is published separately [81].

As confluence and normalisation properties must be proven anew for ES calculi, much research has been devoted to these topics. The (generalised) interpretation method [67, 76] has been used to prove confluence for  $\lambda_{\text{xgc}}$ ,  $\Lambda_{\text{sub}}$ , and  $\lambda_{\text{es}}$  [20, 122, 78]. Tait and Martin-Löf’s technique [8] has been used to prove confluence of  $\lambda_{\text{es}}$  and  $\Lambda_{\text{sub}}$  [78, 81]. Open confluence is an important property if the calculus is to be used to reason about a theorem prover as metavariables allow one to represent proof trees with unknown subtrees [46].

Similarly, established techniques such as the recursive path ordering technique [48, 71] have provided methods of proving PSN for explicit substitution calculi [19]. Other techniques are specific to the calculi such as Bloo and Rose’s use of ‘garbage-free reduction’ [20].

Another method of proving normalisation properties is by introducing a translation from the calculus under study to a second calculus and proving that a weak simulation (which is strong for some subcalculus) exists between the two calculi. Proofs by contradiction can then show that normalisation properties can be reflected back through the simulation. However, this technique is typically of limited application as it requires a second calculus with certain normalisation properties which is operationally close to the first calculus.

In Chapter 9, we present a new method of proving PSN for an explicit substitution calculus with non-local substitution ( $\Lambda_{\text{sub}}$ ) by simulating reduction in a calculus with local substitution.

$\Lambda_{\text{sub}}$  has full composition of substitutions so we therefore require a second calculus with both PSN and FCS. Two such calculi exist to our knowledge and we apply our method (in different ways) to those calculi with positive results. Admittedly, this method is limited at present to the specific case where we apply it but we hope that it can be adapted to other calculi in the future. Specifically, we hope it can be applied to prove normalisation properties of bigraphical encodings of calculi with explicit substitutions by exploiting normalisation properties of the term rewriting systems the Brss are based upon.

**Terminology** (body of substitution, garbage, garbage-free). *The subterm  $u$  in  $t[x/u]$  is called a body of substitution. A body of substitution  $u$  in  $t[x/u]$  is called garbage if  $x \notin \text{FV}(t)$ . We say that  $t$  is garbage-free if it contains no garbage.*

The following terminology may be particular to this work and is used in Chapter 8.

**Terminology** (creates garbage). *We say that a reduction  $(\lambda x.t)u \rightarrow_b t[x/u]$  creates garbage when  $x \notin \text{FV}(t)$ .*

**Notation** (in(side), under a body of substitution). *We say  $v$  is in a body of substitution  $u$  if  $v$  is a subterm of  $u$ . We say  $v$  is under a body of substitution  $u$  if  $v$  is a subterm of  $t$  in  $t[x/u]$ .*

**Definition 2.55** (top-level substitution). *If a substitution  $[x/u]$  in a term  $t$  does not lie inside any other substitution then it is called a top-level substitution. Top-level substitutions may, however, lie under other substitutions.*

## 2.7 Type Systems

We will consider type systems for the  $\lambda$ -calculus in later chapters. In particular, we concentrate on type disciplines of simple types and intersection types. Terms which are typable in the simply typed  $\lambda$ -calculus are strongly normalising [141]; in general, the converse is not true. Intersection types, introduced by Coppo and Dezani-Ciancaglini [40, 41], extend Curry's simply type system with finite polymorphism. Pottinger [129] proved that the set of terms typable under an intersection type system is *exactly* the set of strongly normalising  $\lambda$ -terms.

Given an arbitrary non-empty countable set of *atomic* types  $\mathcal{G}$ , the set of *simple types* is given by the inductive definition

$$\tau ::= G \mid \tau \rightarrow \tau \text{ where } G \in \mathcal{G}.$$

The set of *intersection types* is given by the inductive definition

$$\tau ::= G \mid \tau \rightarrow \tau \mid \tau \wedge \tau \text{ where } G \in \mathcal{G}.$$

Types of the form  $\tau \rightarrow \tau$  and  $\tau \wedge \tau$  are called *function types* and *intersection types* respectively. We let  $G, G'$  range over atomic types,  $\alpha, \alpha'$  range over function types, and  $A, B$ , and  $C$  range over arbitrary types.

An *environment* is a function from a finite set of variables to types. We let  $\Gamma, \Delta$  range over environments. We write  $x : A \in \Gamma$  if  $\Gamma(x) = A$  for some environment  $\Gamma$ . Two environments  $\Gamma$  and  $\Delta$  are said to be *compatible* exactly when they agree on the intersection of their domains. We write  $\Gamma \uplus \Delta$  to denote the *union of compatible contexts*<sup>11</sup>. If  $\text{dom}(\Gamma) \subseteq \text{dom}(\Gamma')$  and both environments are compatible, we overload notation and write  $\Gamma \subseteq \Gamma'$ .

A *typing judgement* is a triple  $\Gamma \vdash t : A$  (where  $t$  is a term) stating that  $t$  has type  $A$  under environment  $\Gamma$ . Typing judgements are derived from a set of *typing rules*. The typable terms of a system are exactly those for which a judgement can be derived using the rules of the system.

The typing rules of the simply typed  $\lambda$ -calculus  $\lambda^\rightarrow$  are presented in Figure 2.6 in the style of Curry. We will consider both an *additive* and a *multiplicative*<sup>12</sup> intersection type discipline. These are respectively presented in Figures 2.7 and 2.8.

We equip the intersection types with a subtyping preorder.

**Definition 2.56.** *The relation  $\ll$  on types is defined by the following axioms and rules.*

1.  $A \ll A$
2.  $A \ll B \ \& \ B \ll C$  implies  $A \ll C$
3.  $A \wedge B \ll A$
4.  $A \ll B \ \& \ A \ll C$  implies  $A \ll B \wedge C$
5.  $A \wedge B \ll B$

<sup>11</sup>This overloads our notation;  $\uplus$  also denotes the union of two sets known to be disjoint. The context will disambiguate this overloading.

<sup>12</sup>These terms are drawn from Linear Logic and originate from a semantic interpretation in coherence spaces [61]. In the context of typing, the multiplicative discipline forbids sharing between the typing environments whereas the additive discipline requires sharing. However, forbidding environments to share is more sensible when dealing with *linear* terms (see Section 7.2.2). We relax the requirement, only ensuring that environments are compatible, as we will apply this discipline to non-linear terms. Note that if  $\Gamma \vdash t : A$  then we respectively have  $\text{dom}(\Gamma) = \text{FV}(t)$  and  $\text{dom}(\Gamma) \supseteq \text{FV}(t)$  in the multiplicative and additive disciplines.

|                                                                                                                  |                                                                                                     |
|------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| $\frac{}{\Gamma, x : A \vdash x : A} \quad (\text{axiom})$                                                       |                                                                                                     |
| $\frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash (t u) : B} \quad (\text{app})$ | $\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B} \quad (\text{abs})$ |

**Figure 2.6:** *A simply typed discipline for the  $\lambda$ -calculus*

|                                                                                                                  |                                                                                                     |
|------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| $\frac{}{\Gamma, x : A \vdash x : A} \quad (\text{axiom})$                                                       |                                                                                                     |
| $\frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash (t u) : B} \quad (\text{app})$ | $\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B} \quad (\text{abs})$ |
| $\frac{\Gamma \vdash t : A \quad \Gamma \vdash t : B}{\Gamma \vdash t : A \wedge B} \quad (\wedge \text{ I})$    | $\frac{\Gamma \vdash t : A_1 \wedge A_2}{\Gamma \vdash t : A_i} \quad (\wedge \text{ E})$           |

**Figure 2.7:** *System  $\text{add}_\lambda$ : An additive intersection type discipline for the  $\lambda$ -calculus*

|                                                                                                               |                                                                                                                                |
|---------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| $\frac{}{x : A \vdash x : A} \quad (\text{axiom})$                                                            | $\frac{\Gamma \vdash t : A \rightarrow B \quad \Delta \vdash u : A}{\Gamma \uplus \Delta \vdash (t u) : B} \quad (\text{app})$ |
| $\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B} \quad (\text{abs}_1)$         | $\frac{\Gamma \vdash t : B \text{ and } x \notin \Gamma}{\Gamma \vdash \lambda x.t : A \rightarrow B} \quad (\text{abs}_2)$    |
| $\frac{\Gamma \vdash t : A \quad \Gamma \vdash t : B}{\Gamma \vdash t : A \wedge B} \quad (\wedge \text{ I})$ | $\frac{\Gamma \vdash t : A_1 \wedge A_2}{\Gamma \vdash t : A_i} \quad (\wedge \text{ E})$                                      |

**Figure 2.8:** *System  $\text{mul}_\lambda$ : A multiplicative intersection type discipline for the  $\lambda$ -calculus*

## Part I

# Kind Sorted Bigraphs

---

## Outline of Part I

We formalise a suggestion by Jensen and Milner to refine the placing structure of bigraphs. This is achieved by a place sorting which we call kind sorting. This refinement generalises the notion of atomic and non-atomic controls, allowing a control to contain a subset of the set of controls. We further enrich placing by adding hidden places and capacities. We show that these variations retain the dynamic theory of pure bigraphs.

We define subcategories of kind sortings. Subcategories are not necessarily sortings with respect to pure bigraphs but we show that subcategories with certain properties retain the dynamic theory. We consider the static theory of pure bigraphs in the sorted setting, identifying how the pure concepts lift to the sorted setting.

Finally, we present some ideas for variants of kind sortings, a link sorting inspired by tile-based games to be used in later examples, and a simple method of combining sortings.

# Chapter 3

## Kind Bigraphs

*In the place where I make no mistakes*

*Waltz #2 (XO) – Elliott Smith*

### 3.1 Introduction

Pure bigraphs treat locality and connectivity as orthogonal concepts, allowing many structures to be represented in bigraphs. However, they are rather free, unrestricted structures themselves. This limits the types of systems that bigraphs can correctly model *e.g.* calculi with types, sorts, or bound variables require certain constraints to be respected with respect to type rules, sortings, and scope rules respectively. Milner notes that “*in significant applications we are quite likely to employ a rich signature*” [110]. To this end, many limitations of pure bigraphs have been addressed in the literature by adding bound variables [74, 106], sorting controls, ports, and names [92, 113], sorting edges [24], sorting by decomposable predicates [16], and polarized links [63]. These enrichments of pure bigraphs are usually related to pure bigraphs via a functor, along which much of the pure theory is shown to be preserved.

One of Jensen and Milner’s suggestions for adding structure to pure bigraphs was to assign a *kind* to each node which stated the controls of nodes that could be contained [74, Section 6]. In this chapter we investigate this suggestion and define what we call *kind sorting* which is a particular place sorting. Our initial definition has been both simplified over time and generalised. We present



the different incarnations here, in increasing generality, for three reasons: it eases the presentation; the level of generality should fit the application; and, most importantly, the level of generality is proportional to the complexity of the axiomatisation in our interpretation.

Fundamentally, kind sorting generalises the notion of atomicity in the place graph structure (the hierarchical tree-like structure) of bigraphs. The controls in a pure bigraphical system are either non-atomic (can contain nodes of any control) or atomic (cannot contain anything). The controls of a kind system – a system under a kind sorting – may contain nodes of some *subset* of the set of controls. Only the place graph is affected and so locality and connectivity remain orthogonal concepts. Once we have introduced these fundamental kind bigraphs, we develop their theory further by first adding internal structure to controls and then by assigning exact, maximum, and minimum capacities to them. These additions allow us to better model structured terms such as encodings of terms of some calculi which have internal structure and capacities *e.g.* in the  $\lambda$ -calculus, a  $\lambda$ -application has a left and right side and each side contains exactly one  $\lambda$ -term. Combined with the fundamental kind sorting, this allows us to model certain typed calculi *e.g.* the simply-typed and intersection typed  $\lambda$ -calculi (see Chapter 11). Other applications involve encoding XML data or models with a sorted hierarchical structure (see the Motivation section below) as bigraphs.

It is not sufficient to simply add structure to pure bigraphs; we must also preserve their rich reactive theory so that we may analyse properties of kind sorting bigraphical reactive systems. In this chapter, we will show that our kind sortings form s-categories. In the following chapters of this part of the dissertation, we will address our hypothesis and prove that sufficient reactive theory is preserved by them.

### 3.1.1 Motivation

To date, various calculi have been modelled with bigraphs: the asynchronous  $\pi$ -calculus [74]; the full  $\pi$ -calculus and ambient calculus [73]; the polyadic  $\pi$ -calculus [24]; arithmetic nets and condition-event Petri nets [92]; finite CCS [113]; and the  $\lambda$ -calculus [111, 63]. Some calculi require extra structure to correctly model terms. For example, the syntax of finite CCS is two-sorted so Milner introduced a specific *place sorting* which models the syntax more correctly than pure bigraphs<sup>1</sup>. Thanks to this place sorting, Milner was able to show that: i) the translation from finite CCS into bigraphs is surjective; ii) the sorting eases the analysis of the labelled transition system by disallowing some labels; and iii) thanks to the sorting, strong bisimilarity of CCS is recovered

<sup>1</sup>We show later that this sorting – homomorphic sorting – is a special case of kind sorting.

in the Brs.

One of our goals is to contribute towards extending bigraph theory such that the syntax of calculi may be correctly modelled in a Brs *i.e.* that bigraphs in the Brs correspond exactly to terms of the calculus up to some equivalence. Towards this, we introduce place sortings which subsume homomorphic sorting and also allow modelling of terms with some notion of capacity or ordering of children. Capacity is important for certain grammars; for example, in the  $\lambda$ -calculus, an application has two children, an abstraction has one. Combined with allowing controls to order their children, adding capacity allows us to model Milner’s ‘multi-nodes’ – controls whose children are partitioned in some order. For example, the  $\mathbf{app}^{A \rightarrow B, A}$  control in Figure 11.10 depicts a control modelling a  $\lambda$  application. The function is stored in the left of the node and the argument in the right. This is an example of a 2-node.

We are also able to show that kind sortings allow some simple type systems to be modelled in bigraphs. In Chapter 11, we type Milner’s bigraphical model of the  $\lambda$ -calculus with simple types and intersection types, such that there is a correspondence between derivations of typed terms and sorted bigraphs.

Another application of bigraphs is in modelling abstract systems, where the controls represent ‘tangible’ objects. For example, models of buildings with mobile entities [13], a model of a printing system with printers, jobs, and job producers with locations [15], or even a model of Conway’s Game of Life [59, 47]. These types of models typically require structure that pure bigraphs do not guarantee. For example, mobile entities should not be able to contain buildings, printers should not be able to contain printers, and in Debois and Damgaard’s model of the Game of Life, each cell must contain exactly eight ‘dots.’ These restrictions are all based on the hierarchy and capacity of the place graph structure. We concentrate on adding these restrictions to bigraphs.

When modelling these abstract systems, the goal is to define Brss where all bigraphs correspond to an agent or context of the model. As with bigraphical encodings of other calculi, this facilitates a correct model and eases the operational analysis by removing contexts/labels which do not make sense according to the model. Kind sorting allows a hierarchy<sup>2</sup> to be defined on the set of controls and respected by the sorted Brs. For example, consider the abstract model depicted in Figure 3.1 representing workers who commute between home and work, accessing locations with keys linked to the locations and commuting with the aid of a travel card. The figure suggests a hierarchy of containment, suggesting which entities may contain which other entities. This hierarchy can be enforced on bigraphs using a (fundamental) kind sorting using the signature:

<sup>2</sup>The word ‘hierarchy’ is misleading as it suggests a non-cyclic directed graph structure. Our definition of kind sortings uses a directed graph structure as opposed to this special case.

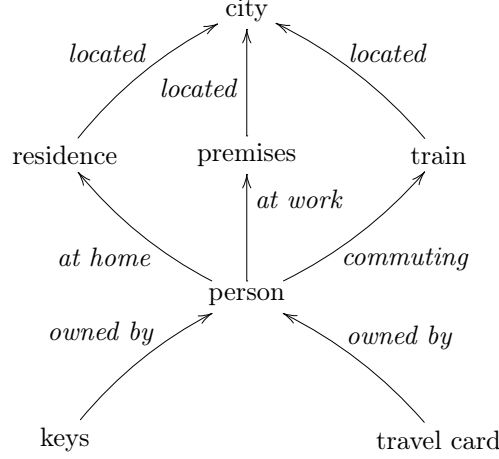


**Figure 3.1:** An abstract model of commuting workers. The arrows describe a containment hierarchy.

|                  |           |                                     |                   |           |                           |
|------------------|-----------|-------------------------------------|-------------------|-----------|---------------------------|
| <b>city</b>      | $\mapsto$ | <b>{premises, residence, train}</b> | <b>person</b>     | $\mapsto$ | <b>{keys, travelcard}</b> |
| <b>premises</b>  | $\mapsto$ | <b>{person, keys, travelcard}</b>   | <b>keys</b>       | $\mapsto$ | $\emptyset$               |
| <b>residence</b> | $\mapsto$ | <b>{person, keys, travelcard}</b>   | <b>travelcard</b> | $\mapsto$ | $\emptyset$               |
| <b>train</b>     | $\mapsto$ | <b>{person, keys, travelcard}</b>   |                   |           |                           |

which states which kinds of control each control can contain (keys and travel cards may be dropped). Hence, each bigraph in a kind sorted Brs based on this signature will respect the hierarchy depicted in the figure.

The long-term aim of bigraphs is to allow a model of large-scale mobile computation *e.g.* as in the Internet and Worldwide Web [74]. Such a model should be able to deal with data transmitted over these infrastructures. It should also allow some notion of reasoning about the specifications of a system. To this end, we will consider the bigraphical modelling of XML data [148] in the disertation as well as the connection between kind sorting and spatial logics for bigraphs<sup>3</sup>. In fact, these topics are related.

Modal logics have already been investigated in the context of analysing mobile computation. Cardelli and Gordon introduced a spatial logic for the ambient calculus in order to express properties of ambient reaction [33, 34]. Caires and Cardelli abstracted this approach and presented a general spatial logic which handles name restriction, fresh name quantification, name hiding, and recursion [26, 27]. These spatial logics are modal in both time and space, both of which are required

<sup>3</sup>The connection with spatial logics is merely foundational; we will discuss axiomatisations of kind sorted bigraphs and discuss the logical expressiveness which arises from sorting bigraph interfaces.

for reasoning about distributed systems, and can express logical statements such as “at some place in process  $P$ , formula  $\mathcal{A}$  holds” or “at some time, there will be a virus at this place.” The modal operators include *everywhere*, *somewhere*, *at all times*, *sometime*. Combined with the classical logical operators, tensor, quantification, hiding, and freshness, the logics are very expressive.

The ambient calculus considers mobility as the change in spatial configurations. In order to correctly reason about bigraphs, we must also consider changes in the connectivity of networks. Conforti, Macedonio, and Sassone have presented BiLog, a spatial logic for pure bigraphs [39]. An interesting note is that their logic is a combination of a link graph logic and a place graph logic, reflecting the orthogonality of these structures.

The sorting of interfaces in kind sorted s-categories admits logical reasoning; we explain this further below. We believe that spatial logics for kind sorted bigraphs could be quite expressive. We do not investigate this claim now but we take the first steps by identifying the new constructions in an axiomatisation of kind sorted bigraphs; the axiomatisation of pure bigraphs [108] is central to BiLog.

Spatial logics are intensional; they describe the internal structures of terms to a fine degree. Caires and Cardelli note that we require this granularity to “*meaningfully describe the distribution of processes and the use of resources over a network.*” The insight that the theory of semi-structured databases [2] and mobile computation (from the perspective of mobile ambients) have technical similarities have led to the use of spatial logics for describing and querying semi-structured data such as (edge-labelled) trees [25, 33, 31, 44], graphs [29], and labelled trees with hidden names [30]. The notion of hiding in this latter work in particular is reminiscent of process calculi and allows more secure descriptions of distributable data such as XML.

Conforti, Macedonio, and Sassone have applied BiLog to describe, query, and reason about bigraphical models of XML data [38]. In their model (and this is common in many of the logics mentioned above), they consider XML data whose document order is irrelevant, presumably as parallel composition is typically associative and commutative. They remark that an extension of bigraphs with a notion of “ordered locality” could be used to model XML data whose document order is relevant. This is our modest addition to the area. Kind sorting can model Milner’s idea of multi-nodes which allow an ordering on the place graph to be encoded. BibTeX entries are another motivating example from the literature [2, 29]. Kind sorting and multi-nodes also allow these to be modelled.

Of course, we are also interested in dynamic properties of kind sorted bigraphs. The simplest property is that the sorting is preserved by reaction. A more interesting property emerges when

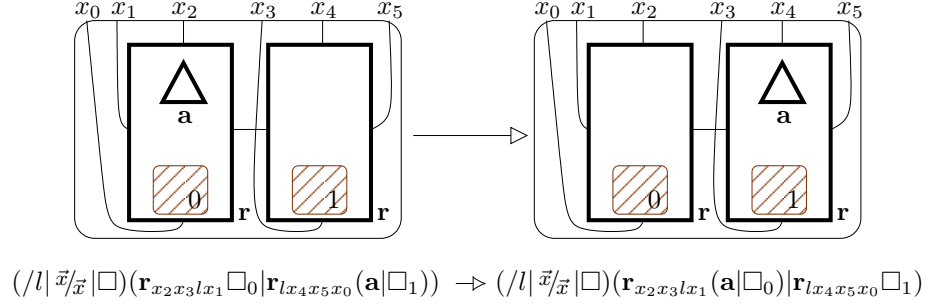


**Figure 3.2:** An agent moves between two rooms

we consider *parametric reaction rules*. These are rules where the subterms are not specified *e.g.* the substitution generation rule  $(\lambda x.t) u \rightarrow_b t[x/u]$  in many explicit substitution calculi does not care about the subterms  $t$  and  $u$  – they are parameters to the rule. These rules can be interesting in kind sorted Brss as we are required to provide a sort for the sites of the rules. However, since the sort of a site specifies which controls the site can contain, it also specifies which controls the site cannot contain; the site guarantees the *absence* of certain controls<sup>4</sup>.

For example, consider a Brs over the signature  $\mathcal{K} = \{\mathbf{a}, \mathbf{e}, \mathbf{r}\}$  where the elements respectively represent agents, enemies, and rooms with exits at cardinal points. Define the kind sorting as

$$\mathbf{r} \mapsto \{\mathbf{a}, \mathbf{e}\} \quad \mathbf{a} \mapsto \emptyset \quad \mathbf{e} \mapsto \emptyset$$

stating that rooms can only contain agents and enemies and agents and enemies cannot contain anything. This signature could be used to model a game played on a map of square tiles<sup>5</sup>. We now add four reaction rules to make agents mobile. Figure 3.2 depicts a ‘move north’ rule. In the pure theory, such rules would let agents move around the board regardless of the contents of the rooms. However, using a sorting we can further specify the rule by sorting the sites of the rule. If we sort the southern site with the sort  $\{\mathbf{a}\}$  then we guarantee that no enemy can be placed in that room. The rule then describes the action ‘an agent may move north if no enemies are present.’ Now consider the rule in Figure 3.3. this states that if an agent occupies the same room as an enemy, the agent is eliminated. We have now described (informally) a bigraphical reactive system (Brs) where agents can roam freely *unless* they run into enemies. If we compare this kind reactive system to the pure reactive system, we see that a kind reactive system can limit or restrict the set of possible reactions that can take place by imposing conditions on a redex that a pure reactive

<sup>4</sup>In some of our kind sortings, a site does not guarantee to *contain* the controls in its sort; we discuss this in the conclusions of this chapter.

<sup>5</sup>This tiling structure can be imposed on bigraphs using the sortings in Section 6.2.2.

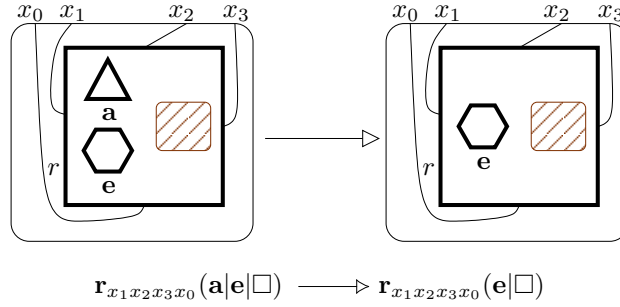


**Figure 3.3:** *An enemy eliminates an agent*

system cannot.

The sorting of sites in parametric reaction rules allows us to specify extra conditions in parametric reaction rules; it allows us to describe preconditions on parameters to the rules concerning the absence of exposed nodes of particular controls. This can sometimes admit a prioritising of the reaction rules (similar to the if/else statement of many programming languages) and can add a level of determinism to the behaviour of the reactive system. Some simple algorithms may also be expressed using this approach. We explore these properties in Chapter 10.

Finally, this sorting of sites and (to a lesser degree) roots may have positive implications for a BiLogic based kind sorted bigraphs. This extra information allows statements like “there is no  $K$ -node below the root” or “this rule will only fire if this parameter has no exposed  $K$ -nodes.” The kind sortings with capacities can express yet more structure. We discuss axiomatisations of kind sorted bigraphs in Chapter 4 as the first step towards applying BiLog to kind sortings.

## 3.2 Place sorting

Many applications require more structure than pure bigraphs can offer. At the end of Chapter 6 we discuss many solutions to this problem in the literature. We are interested in a particular solution here, *place sorting*.

Leifer and Milner introduced the idea of link sortings to add more structure to the link graph structure [92]. Subsequently, Milner and Jensen applied the same idea to the place graph structure [113, 73]. A place-sorting is a sorting discipline which constrains the parent map of a bigraph, admitting only those bigraphs which satisfy the rules of the discipline. The constraints must be preserved by identities, composition, and tensor product; the sorting then describes an s-category of sorted bigraphs. In order to preserve the dynamic theory, the forgetful, sorting functor from

place-sorted bigraphs to pure bigraphs should at least create RPOs.

We let  $\Theta$  denote a non-empty set of *sorts* and let  $\theta$  range over  $\Theta$ .

**Definition 3.1** (place-sorted bigraphs [113]). *An interface with width  $m$  is  $\Theta$ -(place-)sorted if it is enriched by ascribing a sort to each place  $i \in m$ . If  $I$  is place-sorted, its underlying unsorted interface is denoted by  $\mathcal{U}(I)$ .*

$\mathcal{BIG}(\mathcal{K}, \Theta)$  denotes the  $s$ -category over signature  $\mathcal{K}$  in which the objects are place-sorted interfaces, and each arrow  $G : I \rightarrow J$  is a bigraph  $G : \mathcal{U}(I) \rightarrow \mathcal{U}(J)$ . The identities, and composition and tensor product are as in  $\mathcal{BIG}(\mathcal{K})$ , but with sorted interfaces.

We denote place-sorted interfaces as  $\langle m, \vec{\theta}, X \rangle$  where  $\vec{\theta}$  is a vector  $\langle \theta_0, \dots, \theta_{m-1} \rangle$  ascribing a sort  $\theta_i$  to each place  $i \in m$  of the interface. The notation  $\vec{\theta}\vec{\theta}'$  describes the concatenation of vectors  $\vec{\theta}$  and  $\vec{\theta}'$  where  $\langle \theta_0, \dots, \theta_{m-1} \rangle \langle \theta'_0, \dots, \theta'_{n-1} \rangle = \langle \theta_0, \dots, \theta_{m-1}, \theta'_0, \dots, \theta'_{n-1} \rangle$ .

**Definition 3.2** (place-sorting [113]). *A place-sorting is a triple  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  where  $\Phi$  is a condition on  $\Theta$ -sorted bigraphs over  $\mathcal{K}$ . The condition  $\Phi$  must be satisfied by the identities and preserved by composition and tensor product.*

*A bigraph in  $\mathcal{BIG}(\mathcal{K}, \Theta)$  is  $\Sigma$ -(place-)sorted if it satisfies  $\Phi$ . The  $\Sigma$ -sorted bigraphs form a subcategory of  $\mathcal{BIG}(\mathcal{K}, \Theta)$  denoted by  $\mathcal{BIG}(\Sigma)$ . Further, if  $\mathcal{R}$  is a set of  $\Sigma$ -sorted reaction rules then  $\mathcal{BIG}(\Sigma, \mathcal{R})$  is a  $\Sigma$ -sorted Brs.*

Associated with a place-sorting is a forgetful functor  $\mathcal{U} : \mathcal{BIG}(\Sigma) \rightarrow \mathcal{BIG}(\mathcal{K})$  which discards sorts and whose codomain is an  $s$ -category of pure bigraphs.  $\mathcal{U}$  is a sorting functor.

All of our kind sortings are place sortings.

### 3.3 Fundamental kind sorting

In this section, we present our most basic kind sorting. The sorting generalises the notion of atomicity in signatures. In pure signatures, a control may be atomic (can contain nothing) or non-atomic (can contain anything). We alter this to state that a control may contain a subset of the set of controls. This simple generalisation is quite expressive and can encode a level of determinism in the reaction relation (see Chapter 10).

**Definition 3.3** (fundamental kind signature). *A fundamental kind signature  $\{\mathcal{K}, \text{arity}, \text{actv}, \text{kind}\}$  is composed of a set  $\mathcal{K}$  of controls and three maps:*

$$\begin{aligned} \text{arity} : \mathcal{K} &\rightarrow \mathbb{N} & \text{kind} : \mathcal{K} &\rightarrow \mathcal{P}(\mathcal{K}). \\ \text{actv} : \mathcal{K} &\rightarrow \{\text{passive}, \text{active}\} \end{aligned}$$

If  $\text{kind}(K) = \emptyset$  then  $\text{actv}(K) = \text{passive}$ .

The *kind* function maps a control  $K$  to the set of controls that it can contain as *children*<sup>6</sup>, its *kind* or *sort*. If  $\text{kind}(K) = \{L, M\}$  then we allow any node with control  $K$  to only contain nodes of control  $L$  and  $M$  in a kind-sorted bigraph. This generalises pure signatures; atomic (resp. non-atomic) controls  $K$  are now the special case where  $\text{kind}(K) = \emptyset$  (resp.  $\text{kind}(K) = \mathcal{K}$ ). Atomic controls may not be active.

The *kind* function can also be presented as a relation from  $\mathcal{K}$  to  $\mathcal{K}$  or as a directed graph.

**Definition 3.4** (containment relation/graph). *Given a fundamental kind signature over the set  $\mathcal{K}$ , we call the relation  $\triangleright$  defined by  $L \triangleright K$  iff  $L \in \text{kind}(K)$  the containment relation of the signature. We call the directed graph with nodes  $\mathcal{K}$  and  $\triangleright$  as its adjacency relation the containment graph of the signature.*

The containment relation describes the ‘can be contained in’ relation. This is depicted by the containment graph which has an arrow from  $L$  to  $K$  if  $L$ -nodes can be contain in  $K$ -nodes. For example, Figure 3.1 is a (subgraph of) the containment graph of the example in Section 3.1.1.<sup>7</sup>

We will continue to work with functions rather than relations but containment graphs are a useful means of presenting kind signatures.

**Definition 3.5** (fundamental kind sorting<sup>8</sup>). *A place-sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  over a fundamental kind signature  $\mathcal{K}$  is a fundamental kind sorting if  $\Theta = \mathcal{P}(\mathcal{K})$  and  $\Phi$  requires for all bigraphs  $G$  that:*

**K1** if  $p = G(v)$  then  $\text{ctrl}(v) \in \text{kind}(p)$ ;

**K2** if  $p = G(s)$  then  $\text{kind}(s) \subseteq \text{kind}(p)$ ;

**K3** if  $\text{kind}(v) = \emptyset$ ,  $v$  has no children;

where  $p$  is a root or node,  $s$  a site,  $v$  is a node, and the sort/kind of a place  $i$  is written  $\text{kind}(i)$ .

If we instead use an equality in K2, we obtain a simpler theory and axiomatisation but at the expense of expressiveness. Nonetheless, this may be preferable in certain applications so the (more general) kind sorting with rigid capacities later in this chapter allows for this.

<sup>6</sup>We emphasise this point as it can cause confusion. In all the kind sortings presented in this chapter, we only constrain the parent-child relationship in the place graph. The sorting does not constrain *e.g.* grandparent-child relationships *except indirectly i.e.* if control  $K$  can only contain control  $L$  and control  $L$  can not contain control  $M$ , then  $K$  can not be a grandparent of  $M$ .

<sup>7</sup>We omitted the arrows from the keys and travelcard controls to the residence, premises, and train controls.

<sup>8</sup>Søren Debois suggested a simplification of our initial definition which influenced the current one.



**Notation** (kind interface (fundamental / with visibility)). If  $I = \langle m, \vec{\theta}, X \rangle$  is a kind interface of a fundamental kind sorted bigraph then we denote the vector of sorts as  $\langle \theta_0, \dots, \theta_{m-1} \rangle$  where  $\theta_i \subseteq \mathcal{K}, i \in m$  for some signature set  $\mathcal{K}$ . For example, we may write  $I = \langle 2, \langle \{K, L\}, \{L, M\} \rangle, X \rangle$ .

We drop the qualifier ‘fundamental’ for much of the remainder of this section.

A kind sorted bigraph  $G : \langle m, \vec{\theta}, X \rangle \rightarrow \langle n, \vec{\theta}', Y \rangle$  has the same structure as a pure bigraph  $G : \langle m, X \rangle \rightarrow \langle n, Y \rangle$  but it must respect the rules K1-K3 above *i.e.* it respects the ‘hierarchy’<sup>9</sup> defined by the kind signature *i.e.* a root (resp. node) may only be parent of a node when the interface (resp. signature) allows it. A root (resp. node) may only be parent of a site when it can contain at least what the site can contain. The *underlying (pure) bigraph* of  $G$  is denoted by  $G^u$ .

We talk about the sort of a node meaning the sort of the control of that node and treat other properties of nodes similarly. The sort of a node  $v$  is denoted by  $\text{kind}(v)$  and similarly for roots  $r$  and sites  $s$ .

We require places to be sorted so that composition preserves sorting. Composition of kind sorted bigraphs is defined when their underlying bigraphs can be composed.

**Proposition 3.6** (composition respects kind sorting). *If  $A : H \rightarrow I$  and  $B : I \rightarrow J$  are sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.*

*Proof.* Use the definition of composition for pure bigraphs and the fact that  $A$  and  $B$  are sorted. A more general case is proven in Proposition 3.12.  $\square$

**Definition 3.7** (tensor product). *The tensor product of interfaces  $I = \langle m, \vec{\theta}, X \rangle$  and  $J = \langle n, \vec{\theta}', Y \rangle$ , where  $X$  and  $Y$  are disjoint, is  $I \otimes J = \langle m + n, \vec{\theta}\vec{\theta}', X \uplus Y \rangle$ . The tensor product  $G : I \rightarrow J$  of two kind sorted bigraphs  $G_i : I_i \rightarrow J_i$  ( $i = 0, 1$ ) with disjoint node and edge sets is defined when  $I = I_0 \otimes I_1$  and  $J = J_0 \otimes J_1$  are defined, and then  $G^u = G_0^u \otimes G_1^u$ .*

The identities clearly respect the kind rules<sup>10</sup>, as does tensor product; the latter essentially places bigraphs side-by-side and our sorting is localised at the level of roots. Therefore, kind sorting forms wide s-categories with kind interfaces as objects and kind sorted bigraphs as arrows.

**Definition 3.8** (fundamental kind sorted s-category/Brs). *The  $\Sigma$ -sorted (or fundamental kind sorted) bigraphs form a subcategory of  $\mathcal{BIG}(\mathcal{K}, \Phi)$  denoted by  $\mathcal{BIG}(\Sigma)$ . We call  $\mathcal{BIG}(\Sigma)$  a fundamental kind s-category. If  $\mathcal{R}$  is a set of  $\Sigma$ -sorted reaction rules then  $\mathcal{BIG}(\Sigma, \mathcal{R})$  is a  $\Sigma$ -sorted*

<sup>9</sup>See footnote 2 on page 46.

<sup>10</sup>An interesting identity is  $\text{id}_{\langle 1, \emptyset, X \rangle}$ . This is semantically as good as a barren prime bigraph (try composing it with another bigraph) so maybe  $\emptyset$  should not be allowed as an interface sort in the subcategory of hard bigraphs.

(or fundamental kind sorted Brs). The forgetful functor  $\mathcal{U} : \mathsf{BIG}(\Sigma) \rightarrow \mathsf{BIG}(\mathcal{K})$  forgets the sorts of interfaces and is a sorting functor.

**Notation.** We denote full kind sorted  $s$ -categories (either fundamental or those defined in the next sections) as  $\mathsf{BIG}(\Sigma_K)$ .

The faithfulness of  $\mathcal{U}$  simplifies many proofs involving universal constructions (e.g. RPOs). Composition in  $\mathsf{BIG}(\Sigma)$  is defined when the underlying composition in  $\mathsf{BIG}(\mathcal{K})$  is defined.

Kind sortings only affect place graphs; these remain orthogonal structures to link graphs.

### 3.4 Kind sorting with visibility

Our first generalisation of kind sorting adds the notion of *visibility* to controls. A node of a bigraph is said to be *exposed* if it is a child of a root, otherwise *hidden*. We now let kind signatures partition the controls into two sets – *visible* controls and *invisible* controls.

**Definition 3.9** (kind signature with visibility). A kind signature with visibility is composed of a set  $\mathcal{K}$  of controls and five maps:

$$\begin{aligned} \text{arity} : \mathcal{K} &\rightarrow \mathbb{N} & \text{kind}_{\text{inv}} : \mathcal{K}_{\text{vis}} &\rightarrow \mathcal{P}_{\text{fin}}(\mathcal{K}_{\text{inv}}) \uplus \mathcal{K}_{\text{inv}} \rightarrow \{\emptyset\} \\ \text{actv} : \mathcal{K} &\rightarrow \{\text{passive}, \text{active}\} & \text{kind} : \mathcal{K} &\rightarrow \mathcal{P}(\mathcal{K}_{\text{vis}}) \\ \text{vsbl} : \mathcal{K} &\rightarrow \{\text{vis}, \text{inv}\}. \end{aligned}$$

If  $\text{kind}(K) = \emptyset$  then  $\text{actv}(K) = \text{passive}$ .

The function  $\text{vsbl}$  partitions  $\mathcal{K}$  into two sets  $\mathcal{K}_{\text{vis}}$  and  $\mathcal{K}_{\text{inv}}$  of visible and invisible controls. Nodes with invisible controls will not be allowed as children of roots or as elements of interface sorts in the new sorting; they are hidden from the interfaces. This allows us to add some extra structure which is internal to the place graph. We only consider two levels of visibility here since kind sorting only constrains the parent-child relationship. A more expressive  $\text{kind}$  function may admit a generalisation where  $\text{vsbl} : \mathcal{K} \rightarrow \mathbb{N}$  specifies the minimum nesting level at which a node of some control must at least be under a root. The main motivations for introducing invisible controls are to add order to the place graph and to model multi-nodes.

The containment graph we associate with these signatures has labelled edges; as before,  $\mathcal{K}$  is the set of nodes, there is an arrow from  $L$  to  $K$  iff  $L \in \text{kind}_{\text{inv}}(K) \cup \text{kind}(K)$ . We also label an arrow with ‘1’ when the source is an invisible control. This labelling describes that nodes with the target control will contain exactly one node with the invisible control (this is enforced by the sorting below).

This definition of the signature above and sorting below differs from our original version [123]. We now require that a node has exactly one of each kind of invisible control below it, that invisible controls may not parent other invisible controls, and that the set of invisible controls associated with a control is finite. These decisions were taken to ease the axiomatisation of these sorted bigraphs. Otherwise, our requirement that invisible controls cannot be elements of interface sorts complicates the definition of an ion (see Section 4.2); there would be at least countably many ions for each control.

**Convention.** *For all the signatures with invisible controls in this chapter, we implicitly require that a preorder  $\sqsubseteq$  is defined on the invisible elements of the signature set  $\mathcal{K}$  such that a total order is defined on  $\text{kind}_{\text{inv}}(K)$  for all  $K \in \mathcal{K}$ .*

The convention should really be part of the definitions of signatures but it disrupts the presentation and is unimportant for most of the dissertation. It will be required for the axiomatisations of kind bigraphs with invisible controls as it defines a definite order on the sites of ions.

**Notation.** *Let  $p_{K,G}$  denote the number of nodes of control  $K$  under a root/node  $p$  in a bigraph  $G$ .*

**Definition 3.10** (kind sorting with visibility). *A place-sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  over a kind signature  $\mathcal{K}$  with visibility is a kind sorting with visibility if  $\Theta = \mathcal{P}(\mathcal{K}_{\text{vis}})$  and  $\Phi$  requires for all bigraphs  $G$  that:*

**KV1** *if  $p = G(v)$  then  $\text{ctrl}(v) \in \text{kind}(p)$ ;*

**KV2** *if  $p = G(s)$  then  $\text{kind}(s) \subseteq \text{kind}(p)$ ;*

**KV3** *if  $K \in \text{kind}_{\text{inv}}(v)$  then  $v_{K,G} = 1$ ;*

**KV4** *if  $\text{kind}(v) = \emptyset$ ,  $v$  has no children;*

*where  $p$  is a root or node,  $s$  a site, and  $v$  is a node.*

The interfaces of bigraphs are similar to those for a fundamental kind sorting except that the sort of a place will contain no invisible controls. If we ignore condition KV3 then we could instead do away with invisible controls and form a subcategory by restricting the set of interface sorts. However, we prefer our approach here. Firstly, by explicitly stating the invisible controls in the signature, our intentions are clear from the start. Secondly, subcategories of kind sorted bigraphs are not guaranteed to retain the good properties the full s-categories enjoy as we will see in Chapter 5. In that chapter, we prove that the sorting with visibility enjoys good properties. We feel that the above presentation is neater than defining visibility by restricting interface sorts.

Invisible controls now allow considerable expressiveness. We can specify in the signature that a control  $K$  contains exactly a certain number of distinct invisible controls. As invisible controls can not appear in interface sorts, any sorted bigraph containing  $K$ -nodes *must* have the required number of the invisible nodes as children. In essence, the invisible controls are a part of the  $K$ -node as they cannot be separated from it. This allows us to model more complicated controls, one example of which is Milner's multi-nodes.

**Definition 3.11** (multi-node/ $m$ -node). *An  $m$ -node of a kind signature with visibility is a visible control  $K$  where  $\text{kind}(K) = \emptyset$  and  $|\text{kind}_{\text{inv}}(K)| = m$ .*

The idea of multi-nodes, or multi-site controls, for bigraphs was proposed by Milner and inspired our work in Chapter 11. His formal definition [114] and ours were independently derived but share similarities. Both use the notion of invisible controls, controls which are structurally tied to a visible control and which are hidden from outer interfaces. Milner's definition implies a total order on invisible controls which would be important in axiomatisations as it removes any ambiguity about the ordering of sites in ions. We could not require a total order as it would restrict the kind sorts allowed for invisible nodes but we would require a partial order such that any set  $\text{kind}_{\text{inv}}(K), K \in \mathcal{K}_{\text{vis}}$  is totally ordered. The main difference between the definitions is that we allow multi-nodes to have both visible and invisible children. This choice was motivated by the desire to model XML documents with both ordered and unordered children.

**Proposition 3.12** (composition respects kind sorting). *If  $A : H \rightarrow I$  and  $B : I \rightarrow J$  are sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.*

*Proof.* Let  $A$  and  $B$  be sorted with  $\text{cod}(A) = \text{dom}(B)$  and  $B^u \circ A^u$  defined. We must prove the conditions hold for all nodes and roots of  $B \circ A$ .

If  $p$  is a node of  $A$  then the conditions holds as  $A$  is sorted.

Let  $p$  be a node or root of  $B$ . As  $B$  is sorted, KV4 holds immediately and KV3 holds as for all roots  $r <_B p$ , no invisible node is an element of  $\text{kind}(r)$  and  $A$  is sorted. For KV1, let  $v <_{B \circ A} p$ . Either  $v <_B p$  or  $v <_A r, r <_B p$ . In the former case, KV1 holds as  $B$  is sorted. In the latter case, we have  $\text{ctrl}(v) \in \text{kind}(r)$  and  $\text{kind}(r) \subseteq \text{kind}(p)$ , hence  $\text{ctrl}(v) \in \text{kind}(p)$ . For KV2, let  $s <_{B \circ A} p$ . Then  $s <_A r, r <_B p$ . We have  $\text{kind}(s) \subseteq \text{kind}(r) \subseteq \text{kind}(p)$ .  $\square$

The sorting rules are also preserved by identities, composition, and tensor product. Therefore, they form wide s-categories with kind interfaces as objects and kind sorted bigraphs as arrows.

**Definition 3.13** (kind sorted s-category/Brs with visibility). *The  $\Sigma$ -sorted (or kind sorted with visibility) bigraphs form a subcategory of  $\text{BIG}(\mathcal{K}, \Phi)$  denoted by  $\text{BIG}(\Sigma)$ . We call  $\text{BIG}(\Sigma)$  a kind*

s-category with visibility. If  $\mathcal{R}$  is a set of  $\Sigma$ -sorted reaction rules then  $\mathcal{BIG}(\Sigma, \mathcal{R})$  is a  $\Sigma$ -sorted (or kind sorted Brs with visibility). The forgetful functor  $\mathcal{U} : \mathcal{BIG}(\Sigma) \rightarrow \mathcal{BIG}(\mathcal{K})$  is a sorting functor.

Note that the fundamental sorting is a special case where all nodes are visible.

### 3.5 Kind sortings with maximum capacities

In these kind sortings we allow the maximum number of children of a node to be bounded; we both allow controls to contain exactly or at most some number of certain controls and we also specify a guaranteed or maximum control-independent capacity.

**Lemma 3.14.** *Let  $p$  be a node or root of  $B$  and let  $B \circ A$  be defined. Then:*

1.  $p_{K, B \circ A} = p_{K, B} + \sum_{r <_B p} r_{K, A};$
2.  $s <_{B \circ A} p$  iff  $s <_A r$  and  $r <_B p$ .

*Proof.* By the definition of composition of place graphs. □

**Definition 3.15** ( $\mathbb{N}^\odot$ ). We define the set  $\mathbb{N}^\odot = \mathbb{N} \cup \{\odot\}$ . We extend the partial order  $\leq$  by defining  $n \leq \odot, n \in \mathbb{N}^\odot$  and extend the addition operation with  $n + \odot = \odot, n \in \mathbb{N}^\odot$ .

The symbol  $\odot$  represents an arbitrary amount. This will be used to specify that a node of a bigraph may contain finitely many other nodes. Recall that  $\mathbf{Set}(A, B)$  denotes a homset i.e. the set of functions from  $A$  to  $B$  in the category of sets.

**Definition 3.16** (kind signature with rigid capacities). A kind signature over  $\mathcal{K}$  with rigid capacities is composed of a set  $\mathcal{K}$  of controls and five maps:

$$\begin{aligned} \text{arity} : \mathcal{K} &\rightarrow \mathbb{N} & \text{kind}_{\text{inv}} : \mathcal{K}_{\text{vis}} &\rightarrow \mathcal{P}_{\text{fin}}(\mathcal{K}_{\text{inv}}) \uplus \mathcal{K}_{\text{inv}} \rightarrow \{\emptyset\} \\ \text{actv} : \mathcal{K} &\rightarrow \{\text{passive}, \text{active}\} & \text{kind} : \mathcal{K} &\rightarrow \mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^\odot) \\ \text{vsbl} : \mathcal{K} &\rightarrow \{\text{vis}, \text{inv}\} \end{aligned}$$

where the set  $\{K' \mid K' \in \mathcal{K}, \text{kind}(K)(K') > 0, \text{kind}(K)(K') \neq \odot\}$  is finite for all  $K$ . If  $\text{kind}(K) = \emptyset^{\mathcal{K}}$  then  $\text{actv}(K) = \text{passive}$ . We define  $\text{cpc}(K) = \sum_{K' \in \mathcal{K}_{\text{vis}}} \text{kind}(K)(K')$ .

We call  $\text{kind}(K)$  and  $\text{cpc}(K)$  the *kind* of  $K$  and the *capacity* of  $K$  respectively. In our first sorting below, the kind of a control  $K$  will specify exactly how many of each control in  $\mathcal{K}$  a node with control  $K$  may contain in a sorted bigraph. As  $\mathcal{K}$  may be infinite, we require that a control

can only contain exact amounts (i.e. not  $\odot$ ) of controls from a finite subset of  $\mathcal{K}$  as bigraphs are finite structures. The capacity of a control  $K$  will be exactly the sum over  $\mathcal{K}$  of how many of each control  $K$  may contain. We do not include invisible controls in the capacity as their number will be constant.

The idea behind the signature is as follows. It allows us to specify the exact number of  $K'$ -nodes that a  $K$ -node will contain in a ground bigraph, for each  $K'$  in some signature  $\mathcal{K}$ .

**Convention.** When we are discussing the (minimum/exact/maximum) kind of a control or interface sort (see below), we typically ignore elements of  $\mathcal{K}$  whose image is 0.

The containment graph we associate with these signatures has labelled edges specifying how many of each control a node may contain. For example, Figure 3.4(b) on page 64 depicts that  $K$ -nodes contain exactly three  $L$ -nodes and exactly two  $M$ -nodes in ground bigraphs. This will be enforced by the first sorting below.

**Definition 3.17** (rigid interface sorts). *To every kind signature over  $\mathcal{K}$  with rigid capacities, we associate a set of rigid capacity interface sorts defined as the largest subset of  $\mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^\odot)$  such that for all elements  $\theta$ , the set  $\{K' \mid K' \in \mathcal{K}, \theta(K') > 0, \theta(K') \neq \odot\}$  is finite. If a place  $r$  is assigned a sort  $\theta \in \mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^\odot)$ , we write  $\text{kind}(r)$  for  $\theta$  and define  $\text{cpc}(r) = \sum_{K \in \mathcal{K}_{\text{vis}}} \text{kind}(r)(K)$ .*

**Definition 3.18** (kind sorting with rigid capacities). *A place-sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  over a kind signature over  $\mathcal{K}$  with rigid capacities is a kind sorting with rigid capacities if  $\Theta$  is the set of rigid capacity interface sorts and  $\Phi$  requires for all bigraphs  $G$  that:*

$$\mathbf{KR1} \quad \text{kind}(p)(K) = p_{K,G} + \sum_{s <_G p} \text{kind}(s)(K) \text{ for all } K \in \mathcal{K}_{\text{vis}} \text{ when } \text{kind}(p)(K) \neq \odot;$$

$$\mathbf{KR2} \quad \text{if } K \in \text{kind}_{\text{inv}}(v) \text{ then } v_{K,G} = 1;$$

$$\mathbf{KR3} \quad \text{if } \text{cpc}(p) = 0, p \text{ has no visible children};$$

where  $p$  is a root or node,  $v$  is a node, and  $s$  is a site.

The conditions ensure that for all ground bigraphs, each root and node contains exactly what it should according to its kind. For non-ground bigraphs, the conditions ensure similar properties except that we now consider the sorts of sites. The final condition is necessary only because we allow interface places to have zero capacities. This allowance does not seem to make sense semantically for this sorting<sup>11</sup> but we allow it in all the following sortings in case it may be required.

<sup>11</sup>There seems no reason to allow an empty root in a bigraph with zero outer capacity. An empty root may be required in a redex (resp. reactum) in a reaction rule but in bigraph theory it must then be sorted according to reactum (resp. redex) so that the outer interface is preserved through reaction. In this case, we would probably require that the root had a capacity of  $\odot$ .

**Definition 3.19** (kind interface (with rigid capacities)). An interface  $I = \langle m, \vec{\theta}, X \rangle$  where  $\vec{\theta}$  consists of rigid interface sorts is a kind interface with rigid capacities.

**Notation.** If  $\theta$  is a rigid interface sort over some signature  $\mathcal{K}$  then we denote  $\theta$  as a tuple of elements  $\theta(K)$  for all  $K \in \mathcal{K}_{\text{vis}}$  where  $\theta(K) > 0$ . For example, we may write  $\theta$  as  $(3K, 2L)$  if  $\theta(K) = 3$  and  $\theta(L) = 2$ . We extend this notation to interfaces e.g. writing  $I = \langle 2, \langle (3K, 2L), (4L, 1M) \rangle, X \rangle$ .

This sorting can also model multi-nodes.

**Definition 3.20** (multi-node/ $m$ -node). An  $m$ -node of a kind signature with rigid capacities is a visible control  $K$  where  $\text{kind}(K)(K') = 0$  for all (visible) controls  $K'$  and  $|\text{kind}_{\text{inv}}(K)| = m$ .

**Proposition 3.21** (composition respects kind sorting with rigid capacities). If  $A : H \rightarrow I$  and  $B : I \rightarrow J$  are sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.

*Proof.* Let  $A$  and  $B$  be sorted with  $\text{cod}(A) = \text{dom}(B)$  and  $B^u \circ A^u$  defined. We must prove the conditions hold for all nodes and roots of  $B \circ A$ .

If  $p$  is a node of  $A$  then the condition holds as  $A$  is sorted.

Let  $p$  be a node or root of  $B$ . As  $B$  is sorted, KR3 holds immediately and KR2 holds as for all roots  $r <_B p$ , no invisible node is an element of  $\text{kind}(r)$  and  $A$  is sorted. For KR1, we assume w.l.o.g. that  $\text{kind}(p)(K) \neq \odot$ . Thus, for all places  $r <_B p$ ,  $\text{kind}(r)(K) \neq \odot$ . We have:

$$\begin{aligned}
& \langle A \text{ is sorted} \rangle \\
\Rightarrow & \forall_{K \in \mathcal{K}_{\text{vis}}}. \forall_{r <_B p}. (\text{kind}(r)(K) = r_{K,A} + \sum_{s <_A r} \text{kind}(s)(K)) \\
\Rightarrow & \forall_{K \in \mathcal{K}_{\text{vis}}}. \left( \sum_{r <_B p} \text{kind}(r)(K) = \sum_{r <_B p} r_{K,A} + \sum_{r <_B p} \sum_{s <_A r} \text{kind}(s)(K) \right) \\
\Rightarrow & \forall_{K \in \mathcal{K}_{\text{vis}}}. \left( p_{K,B} + \sum_{r <_B p} \text{kind}(r)(K) = p_{K,B} + \sum_{r <_B p} r_{K,A} + \sum_{s <_{B \circ A} p} \text{kind}(s)(K) \right) \\
& \langle B \text{ is sorted, } \text{kind}(p)(K) \neq \odot, \text{ Lemma 3.14} \rangle \\
\Rightarrow & \forall_{K \in \mathcal{K}_{\text{vis}}}. (\text{kind}(p)(K) = p_{K,B} + \sum_{r <_B p} \text{kind}(r)(K) = p_{K,B \circ A} + \sum_{s <_{B \circ A} p} \text{kind}(s)(K))
\end{aligned}$$

□

This sorting is called *rigid* since we specify exactly how many of each control a control  $K$  may contain. We can relax this property by instead adding a maximum capacity to the signature and then relaxing the equality in KR1 above to an inequality. A number of sortings arise from this. We present two of those now.

**Definition 3.22** (kind signature with maximum capacities). *A kind signature over  $\mathcal{K}$  with maximum capacities is composed of a set  $\mathcal{K}$  of controls and six maps:*

$$\begin{aligned} \text{arity} : \mathcal{K} &\rightarrow \mathbb{N} & \text{kind}_{\text{inv}} : \mathcal{K}_{\text{vis}} &\rightarrow \mathcal{P}_{\text{fin}}(\mathcal{K}_{\text{inv}}) \uplus \mathcal{K}_{\text{inv}} \rightarrow \{\emptyset\} \\ \text{actv} : \mathcal{K} &\rightarrow \{\text{passive}, \text{active}\} & \text{kind} : \mathcal{K} &\rightarrow \mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^{\odot}) \\ \text{vsbl} : \mathcal{K} &\rightarrow \{\text{vis}, \text{inv}\} & \text{cpc} : \mathcal{K} &\rightarrow \mathbb{N}^{\odot} \end{aligned}$$

where: the set  $\{K' \mid K' \in \mathcal{K}, \text{kind}(K)(K') > 0, \text{kind}(K)(K') \neq \odot\}$  is finite for all  $K$ ;  $\text{cpc}(K) \leq \sum_{K' \in \mathcal{K}_{\text{vis}}} \text{kind}(K)(K')$ ; if  $\sum_{K' \in \mathcal{K}_{\text{vis}}} \text{kind}(K)(K') > 0$  then  $\text{cpc}(K) > 0$ ; and if  $\text{kind}(K) = \emptyset^{\mathcal{K}}$  then  $\text{actv}(K) = \text{passive}$ .

The capacity is now specified separately from the *kind* function. The capacity of a control  $K$  will specify how many nodes a node of control  $K$  may at most contain (in a *ground* bigraph; in non-ground bigraphs we must consider the sorts of sites under the node). The conditions on *cpc* and *kind* ensure that we only define sensible signatures.

The containment graph we associate with these signatures has both labelled edges specifying capacity and decorated controls. For example, Figure 3.4(c) on page 64 depicts that  $K$ -nodes can contain at most three  $L$ -nodes and at most two  $M$ -nodes in ground bigraphs.  $K$  also has a (maximum) capacity of four. We do not decorate atomic controls *e.g.*  $L$  and  $M$  in the figure.

**Definition 3.23** (maximum capacity interface sorts). *To every kind signature over  $\mathcal{K}$  with maximum capacities, we associate a set of maximum capacity interface sorts defined as the largest subset of  $(\mathbb{N}^{\odot} \times \mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^{\odot}))$  such that for all elements we have  $\pi_1(\theta) \leq \sum_{K \in \mathcal{K}_{\text{vis}}} \pi_2(\theta)(K)$  and if  $\sum_{K \in \mathcal{K}_{\text{vis}}} \pi_2(\theta)(K) > 0$  then  $\pi_1(\theta) > 0$ . If a place  $r$  is assigned a sort  $\theta \in (\mathbb{N}^{\odot} \times \mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^{\odot}))$  then we define  $\text{cpc}(r) = \pi_1(\theta)$  and  $\text{kind}(r) = \pi_2(\theta)$ .*

In the next sorting, nodes and roots have a maximum capacity which must again be filled exactly but this time the *kind* constraints are looser.

**Definition 3.24** (kind sorting with semi-rigid maximum capacities). *A place-sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  over a kind signature over  $\mathcal{K}$  with maximum capacities is a kind sorting with semi-rigid maximum capacities if  $\Theta$  is the set of maximum capacity interface sorts and  $\Phi$  requires for all bigraphs  $G$  that:*

$$\mathbf{KSM1} \quad \text{cpc}(p) = \sum_{K \in \mathcal{K}_{\text{vis}}} p_{K,G} + \sum_{s <_{GP} p} \text{cpc}(s) \text{ when } \text{cpc}(p) \neq \odot;$$

$$\mathbf{KSM2} \quad \text{kind}(p)(K) \geq p_{K,G} + \sum_{s <_{GP} p} \text{kind}(s)(K) \text{ for all } K \in \mathcal{K}_{\text{vis}} \text{ when } \text{kind}(p)(K) \neq \odot;$$



**KSM3** if  $K \in \text{kind}_{\text{inv}}(v)$  then  $v_{K,G} = 1$ ;

**KSM4** if  $\text{cpc}(p) = 0$ ,  $p$  has no visible children;

where  $p$  is a root or node,  $v$  is a node, and  $s$  is a site.

**Proposition 3.25** (composition respects kind sorting with semi-rigid maximum capacities). *If  $A : H \rightarrow I$  and  $B : I \rightarrow J$  are sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.*

*Proof.* Let  $A$  and  $B$  be sorted with  $\text{cod}(A) = \text{dom}(B)$  and  $B^u \circ A^u$  defined. We must prove the conditions hold for all nodes and roots of  $B \circ A$ .

If  $p$  is a node of  $A$  then the condition holds as  $A$  is sorted.

Let  $p$  be a node or root of  $B$ . As  $B$  is sorted, KSM4 holds immediately and KSM3 holds as for all roots  $r <_B p$ , no invisible node is an element of  $\text{kind}(r)$  and  $A$  is sorted. For KSM2, the proof is similar to that in Proposition 3.21. For KSM1, we assume w.l.o.g. that  $\text{cpc}(p) \neq \odot$ . Thus, for all places  $r <_B p$ ,  $\text{cpc}(r) \neq \odot$ . We have:

$$\begin{aligned}
& \langle A \text{ is sorted} \rangle \\
\Rightarrow & \quad \forall_{r <_B p} \left( \text{cpc}(r) = \sum_{K \in \mathcal{K}_{\text{vis}}} r_{K,A} + \sum_{s <_A r} \text{cpc}(s) \right) \\
\Rightarrow & \quad \sum_{r <_B p} \text{cpc}(r) = \sum_{K \in \mathcal{K}_{\text{vis}}} \sum_{r <_B p} r_{K,A} + \sum_{r <_B p} \sum_{s <_A r} \text{cpc}(s) \\
\Rightarrow & \quad \sum_{K \in \mathcal{K}_{\text{vis}}} p_{K,B} + \sum_{r <_B p} \text{cpc}(r) = \sum_{K \in \mathcal{K}_{\text{vis}}} \left( p_{K,B} + \sum_{r <_B p} r_{K,A} \right) + \sum_{s <_{B \circ A} p} \text{cpc}(s) \\
& \quad \langle B \text{ is sorted, Lemma 3.14} \rangle \\
\Rightarrow & \quad \text{cpc}(p) = \sum_{K \in \mathcal{K}_{\text{vis}}} p_{K,B} + \sum_{r <_B p} \text{cpc}(r) = \sum_{K \in \mathcal{K}_{\text{vis}}} p_{K,B \circ A} + \sum_{s <_{B \circ A} p} \text{cpc}(s)
\end{aligned}$$

□

In the last of these sortings, the capacity is a maximum which cannot be exceeded but does not have to be filled.

**Definition 3.26** (kind sorting with maximum capacities). *A place-sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  over a kind signature over  $\mathcal{K}$  with maximum capacities is a kind sorting with maximum capacities if  $\Theta$  is the set of maximum capacity interface sorts and  $\Phi$  requires for all bigraphs  $G$  that:*

**KM1**  $\text{cpc}(p) \geq \sum_{K \in \mathcal{K}_{\text{vis}}} p_{K,G} + \sum_{s <_G p} \text{cpc}(s)$  when  $\text{cpc}(p) \neq \odot$ ;

**KM2**  $\text{kind}(p)(K) \geq p_{K,G} + \sum_{s <_G p} \text{kind}(s)(K)$  for all  $K \in \mathcal{K}_{\text{vis}}$  when  $\text{kind}(p)(K) \neq \odot$ ;

**KM3** if  $K \in \text{kind}_{\text{inv}}(v)$  then  $v_{K,G} = 1$ ;

**KM4** if  $\text{cpc}(p) = 0$ ,  $p$  has no visible children;

where  $p$  is a root or node,  $v$  is a node, and  $s$  is a site.

**Definition 3.27** (kind interface (with (semi-rigid) maximum capacities)). *An interface  $I = \langle m, \vec{\theta}, X \rangle$  where  $\vec{\theta}$  consists of maximum capacity interface sorts is a kind interface with (semi-rigid) maximum capacities.*

**Notation.** *If  $\theta$  is a maximum capacity interface sort over some signature  $\mathcal{K}$  then we denote  $\theta$  as a tuple of elements  $\pi_2(\theta)(K)$  for all  $K \in \mathcal{K}_{\text{vis}}$  where  $\pi_2(\theta)K > 0$  and superscript this tuple with the capacity of  $\theta$ . For example, we may write  $\theta$  as  $(3K, 2L)^3$  if  $\pi_2(\theta)(K) = 3$ ,  $\pi_2(\theta)(L) = 2$ , and  $\pi_1(\theta) = 3$ . We extend this notation to interfaces e.g. writing  $I = \langle 2, \langle (3K, 2L)^3, (4L, 1M)^5 \rangle, X \rangle$ .*

**Proposition 3.28** (composition respects kind sorting with maximum capacities). *If  $A : H \rightarrow I$  and  $B : I \rightarrow J$  are sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.*

*Proof.* The proof is similar to Proposition 3.25. □

The three sortings above are preserved by identities, tensor product, and composition. Thus, they form wide s-categories with kind interfaces<sup>12</sup> as objects and kind sorted bigraphs as arrows.

**Definition 3.29** (kind sorted s-category/Brs with (rigid/semi-rigid) maximum capacities). *The  $\Sigma$ -sorted (or kind sorted with (rigid/semi-rigid) maximum capacities) bigraphs form a subcategory of  $\text{BIG}(\mathcal{K}, \Phi)$  denoted by  $\text{BIG}(\Sigma)$ . We call  $\text{BIG}(\Sigma)$  a kind s-category with (rigid/semi-rigid) maximum capacities. If  $\mathcal{R}$  is a set of  $\Sigma$ -sorted reaction rules then  $\text{BIG}(\Sigma, \mathcal{R})$  is a  $\Sigma$ -sorted (or kind sorted Brs with (rigid/semi-rigid) maximum capacities). The forgetful functor  $\mathcal{U} : \text{BIG}(\Sigma) \rightarrow \text{BIG}(\mathcal{K})$  is a sorting functor.*

Kind sortings with visibility are a special case of the three sortings above where  $\text{kind}(K)(K')$  is either 0 or  $\odot$  for all allowed pairs of controls  $K, K'$ ,  $\text{cpc}(K)$  is either 0 or  $\odot$ , and the set of interface sorts satisfies  $\pi_1 \in \{0, \odot\}$  and  $\pi_2(K) \in \{0, \odot\}$  for all controls  $K$ .

In the three sortings above, we have chosen to impose the same conditions on roots and nodes of bigraphs. This was a choice. We could have relaxed the kind sorting with rigid capacities by allowing the kind of nodes to be a maximum rather than an exact amount *i.e.* by defining  $\text{kind}(v)(K) \geq v_{K,G} + \sum_{s <_G v} \text{kind}(s)(K)$ , for all  $K \in \mathcal{K}_{\text{vis}}$ , but still required that the kind of roots specified exact amounts. We do not consider all these possibilities here as we feel that we

<sup>12</sup>Triples  $\langle m, \theta, X \rangle$  where the interfaces sort  $\theta$  depends on the sorting used.

have introduced sufficient examples; proofs involving those other sortings should proceed similarly. When we define conditions on nodes and roots separately like this, two properties of the sortings are revealed: i) the intention of the sortings is expressed by the conditions on controls; and ii) by restricting the conditions on roots, we are defining a subcategory. For kind sortings, if we restrict an inequality to an equality in a condition on roots then the resulting subcategory consists of opcartesian bigraphs (see Section 4.1). In Section 4.3, we define various subcategories of kind sorted bigraphs which consist only of opcartesian bigraphs.

### 3.6 Kind sorting with min-max capacities

Our last generalisation allows the number of children of a node to have lower bounds as well. We allow controls to contain at least some number of certain controls and at most some other number. They are also assigned minimum and maximum capacities. These are quite flexible properties. They allow us to properly model ordered tree structures in bigraphs using multi-nodes, an immediate application of which would be to model XML data whose document order is relevant. They also allows us to better model the  $\lambda$ -calculus and other calculi. However, while allowing both upper and lower bounds on capacities adds a lot of expressiveness, our approach here will lead to complicated axiomatisations for certain signatures. We will explain this further in Chapter 4.

**Definition 3.30** ( $\mathbb{N}^*$ ). *We define the set  $\mathbb{N}^* = \mathbb{N} \cup \{\star\}$ . We extend the partial order  $\leq$  by defining  $\star \leq \star$  and extend the addition operation with  $n + \star = \star, n \in \mathbb{N}^*$ .*

The symbol  $\star$  represents “arbitrarily many.” By our definition,  $\leq$  is no longer a total order which will affect our equations below.

**Definition 3.31** (kind signature with rigid min-max capacities). *A kind signature over  $\mathcal{K}$  with rigid min-max capacities is composed of a set  $\mathcal{K}$  of controls and eight maps:*

$$\begin{array}{ll} \text{arity} : \mathcal{K} \rightarrow \mathbb{N} & \text{cpc}_{\min} : \mathcal{K} \rightarrow \mathbb{N}^* \\ \text{actv} : \mathcal{K} \rightarrow \{\text{passive}, \text{active}\} & \text{cpc}_{\max} : \mathcal{K} \rightarrow \mathbb{N}^* \\ \text{vsbl} : \mathcal{K} \rightarrow \{\text{vis}, \text{inv}\} & \text{kind}_{\min} : \mathcal{K} \rightarrow \mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^*) \\ \text{kind}_{\text{inv}} : \mathcal{K}_{\text{vis}} \rightarrow \mathcal{P}_{\text{fin}}(\mathcal{K}_{\text{inv}}) \uplus \mathcal{K}_{\text{inv}} \rightarrow \{\emptyset\} & \text{kind}_{\max} : \mathcal{K} \rightarrow \mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^*) \end{array}$$

where: the set  $\{K' \mid K' \in \mathcal{K}, \text{kind}_{\min}(K)(K') > 0\}$  is finite for all  $K$ ; if  $\text{kind}_{\max}(K) = \emptyset_{\text{vis}}^{\mathcal{K}}$  then  $\text{actv}(K) = \text{passive}$ ;  $\text{kind}_{\min}(K)(K') \leq \text{kind}_{\max}(K)(K')$  for all  $K, K' \in \mathcal{K}$ ; and for all  $K \in \mathcal{K}$ ,

$$\sum_{K' \in \mathcal{K}_{\text{vis}}} \text{kind}_{\min}(K)(K') \leq \text{cpc}_{\min}(K) \leq \text{cpc}_{\max}(K) \leq \sum_{K' \in \mathcal{K}_{\text{vis}}} \text{kind}_{\max}(K)(K').$$

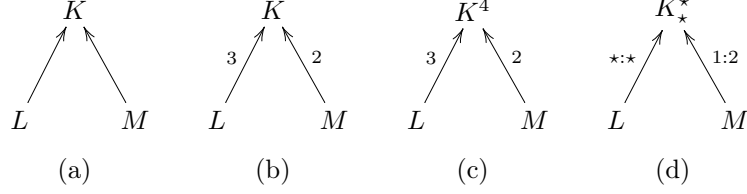


**Figure 3.4:** Containment graphs for a (a) fundamental signature, (b) signature with rigid capacities, (c) signature with (semi-rigid) maximum capacities, (d) signature with rigid min-max capacities.

This signature is similar to Definition 3.16 except we can now also specify a lower bound for both kinds and capacities. The minimum capacity of  $K$  is the number of arbitrary nodes that a  $K$ -node *must at least* contain. If  $\text{kind}_{\min}(K)(K') = \star$  for some  $K'$  then we must have  $\text{cpc}_{\min}(K) = \text{cpc}_{\max}(K) = \text{kind}_{\max}(K)(K') = \star$ . Likewise, if  $\text{cpc}_{\min}(K)$  or  $\text{cpc}_{\max}$  equals  $\star$  then the other must and so must  $\text{kind}_{\min}(K)(K')$  and  $\text{kind}_{\max}(K)(K')$  for some  $K'$ . Despite this, we allow  $\star$  as it is both required so that the signature generalises our previous definitions and attractive for modelling.

However, requiring that nodes *must* contain at least some number of nodes will define s-categories where ground bigraphs cannot exist. For example, let  $\mathcal{K} = \{K\}$ ,  $\text{vsbl}(K) = \text{vis}$ , and  $\text{kind}_{\min}(K)(K) = 1$ . A  $K$ -node must contain at least one  $K$ -node which must itself contain at least one  $K$ -node *et cetera*. Such signatures preclude ground bigraphs which are crucial to the reaction relation. We must prohibit these dangerous cycles.

**Definition 3.32** (cyclic min-max signature). *To every kind signature over  $\mathcal{K}$  with rigid min-max capacities, we associate a directed graph where the set of nodes is  $\mathcal{K}$  and there exists exactly one edge between nodes  $K$  and  $K'$  exactly when  $\text{kind}_{\min}(K)(K') > 0$ . We say that the signature is a cyclic min-max signature if this graph contains cycles.*

From now on, we assume that all min-max signatures are not cyclic.

The containment graph we associate with these signatures has labelled edges specifying min-max capacities and decorated controls. For example, Figure 3.4(d) depicts that in a ground bigraph, a  $K$ -node can contain at least one  $M$ -node and at most two and also can contain arbitrarily many  $L$ -nodes. Further, the  $K$ -node has an arbitrary minimum and maximum capacity. We omit decorations on atomic nodes and in the sequel we will also omit decorations of zero and double decorations of  $\star$  so that  $K_0^3$  will be written  $K^3$  and  $K_\star^\star$  will be written  $K^\star$ .

**Definition 3.33** (rigid min-max interface sorts). *To every kind signature over  $\mathcal{K}$  with rigid min-max capacities, we associate a set of rigid min-max capacity interface sorts defined as the largest subset of  $\mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^*)$  such that for all elements  $\theta$ , the set  $\{K' \mid K' \in \mathcal{K}, \theta(K') > 0, \theta(K') \neq \star\}$  is finite. If a place  $r$  is assigned a sort  $\theta \in \Theta_{\mathcal{K}}$ , we write  $\text{kind}(r)$  for  $\theta$  and define  $\text{cpc}(r) = \sum_{K \in \mathcal{K}_{\text{vis}}} \text{kind}(r)(K)$ .*

The sorting we consider below requires interface places to exactly specify how many of each control lies below them.

**Definition 3.34** (kind sorting with rigid min-max capacities). *A place-sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  over a kind signature over  $\mathcal{K}$  with rigid min-max capacities is a kind sorting with rigid min-max capacities if  $\Theta = \mathbf{Set}(\mathcal{K}_{\text{vis}}, \mathbb{N}^*)$  and  $\Phi$  requires for all bigraphs  $G$  that:*

$$\mathbf{KMM1} \quad \text{kind}(r)(k) = r_{K,G} + \sum_{s <_G r} \text{kind}(s)(k) \text{ for all } K \in \mathcal{K}_{\text{vis}} \text{ when } \text{kind}(r)(K) \neq \star;$$

$$\mathbf{KMM2} \quad \text{cpc}_{\min}(v) \leq \sum_{K \in \mathcal{K}_{\text{vis}}} \left( v_{K,G} + \sum_{s <_G v} \text{kind}(s)(K) \right) \text{ when } \text{cpc}_{\min}(v) \neq \star;$$

$$\mathbf{KMM3} \quad \text{cpc}_{\max}(v) \geq \sum_{K \in \mathcal{K}_{\text{vis}}} \left( v_{K,G} + \sum_{s <_G v} \text{kind}(s)(K) \right) \text{ when } \text{cpc}_{\max}(v) \neq \star;$$

$$\mathbf{KMM4} \quad \text{kind}_{\min}(v)(K) \leq v_{K,G} + \sum_{s <_G v} \text{kind}(s)(K) \text{ for all } K \in \mathcal{K}_{\text{vis}} \text{ when } \text{kind}_{\min}(v)(K) \neq \star;$$

$$\mathbf{KMM5} \quad \text{kind}_{\max}(v)(K) \geq v_{K,G} + \sum_{s <_G v} \text{kind}(s)(K) \text{ for all } K \in \mathcal{K}_{\text{vis}} \text{ when } \text{kind}_{\max}(v)(K) \neq \star;$$

$$\mathbf{KMM6} \quad \text{if } K \in \text{kind}_{\text{inv}}(v) \text{ then } v_{K,G} = 1;$$

$$\mathbf{KMM7} \quad \text{if } \text{cpc}(p) = 0, p \text{ has no visible children};$$

where  $r$  is a root,  $v$  is a node, and  $s$  is a site.

The conditions ensure that for all ground bigraphs: i) each root contains exactly what it should according to its kind; ii) the number of nodes directly under a control does not exceed its maximum capacity and is not less than its minimum capacity; iii) the number of  $K$ -nodes directly under a control does not exceed the maximum allowed and is not less than the minimum allowed. For non-ground bigraphs, the conditions ensure similar properties except that we now consider the sorts of sites.

It is important to note that if a root  $r$  is the parent of a site  $s$  where  $\text{kind}(s)(K) = \star$  then to satisfy the conditions we must have  $\text{kind}(r)(K) = \star$ . If instead a node  $v$  parents this site, we must have  $\text{cpc}_{\min}(v) = \text{cpc}_{\max}(v) = \text{kind}_{\min}(v)(K) = \text{kind}_{\max}(v)(K) = \star$ .

The interfaces of bigraphs are defined as for kind bigraphs with rigid capacities.

**Definition 3.35** (multi-node/ $m$ -node). *An  $m$ -node of a kind signature with rigid min-max capacities is a visible control  $K$  where  $\text{kind}_{\max}(K) = \text{kind}_{\max}(K) = 0$  for all (visible) controls and  $|\text{kind}_{\text{inv}}(K) = m|$ .*

**Proposition 3.36** (composition respects kind sorting with rigid min-max capacities). *If  $A : H \rightarrow I$  and  $B : I \rightarrow J$  are sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.*

*Proof.* Let  $A$  and  $B$  be sorted with  $\text{cod}(A) = \text{dom}(B)$  and  $B^u \circ A^u$  defined. We must prove the conditions hold for all nodes and roots of  $B \circ A$ .

If  $p$  is a node of  $A$  then the conditions holds as  $A$  is sorted.

Let  $t$  be a root of  $B$ . For KMM1, we assume w.l.o.g. that  $\text{kind}(t)(K) \neq \star$ . Thus, for all places  $r <_B t$ ,  $\text{kind}(r)(K) \neq \star$ . We have:

$$\begin{aligned}
& \langle A \text{ is sorted} \rangle \\
\Rightarrow & \forall_{K \in \mathcal{K}_{\text{vis}}}. \forall_{r <_B t}. (\text{kind}(r)(K) = r_{K,A} + \sum_{s <_A r} \text{kind}(s)(K)) \\
\Rightarrow & \forall_{K \in \mathcal{K}_{\text{vis}}}. (\sum_{r <_B t} \text{kind}(r)(K) = \sum_{r <_B t} r_{K,A} + \sum_{r <_B t} \sum_{s <_A r} \text{kind}(s)(K)) \\
\Rightarrow & \forall_{K \in \mathcal{K}_{\text{vis}}}. (t_{K,B} + \sum_{r <_B t} \text{kind}(r)(K) = t_{K,B} + \sum_{r <_B t} r_{K,A} + \sum_{s <_{B \circ A} t} \text{kind}(s)(K)) \\
& \langle B \text{ is sorted, } \text{kind}(t)(K) \neq \star, \text{ Lemma 3.14} \rangle \\
\Rightarrow & \forall_{K \in \mathcal{K}_{\text{vis}}}. (\text{kind}(t)(K) = t_{K,B} + \sum_{r <_B t} \text{kind}(r)(K) = t_{K,B \circ A} + \sum_{s <_{B \circ A} t} \text{kind}(s)(K))
\end{aligned}$$

KMM4 and KMM5 are similarly shown to hold for nodes  $v$ .

Let  $v$  be a node of  $B$ . As  $B$  is sorted, KMM7 holds immediately and KMM6 holds as for all roots  $r <_B t$ , no invisible node is an element of  $\text{kind}(r)$  and  $A$  is sorted. For KMM2, we assume w.l.o.g. that  $\text{cpc}_{\min}(v) \neq \star$ . Thus, for all places  $r <_B v$ ,  $\text{kind}(r)(K) \neq \star$  for all  $K \in \mathcal{K}$ . We have:

$$\begin{aligned}
& \langle A \text{ is sorted} \rangle \\
\Rightarrow & \forall_{K \in \mathcal{K}_{\text{vis}}}. \forall_{r <_B v}. (\text{kind}(r)(K) = r_{K,A} + \sum_{s <_A r} \text{kind}(s)(K)) \\
\Rightarrow & \sum_{K \in \mathcal{K}_{\text{vis}}} \sum_{r <_B v} \text{kind}(r)(K) = \sum_{K \in \mathcal{K}_{\text{vis}}} (\sum_{r <_B v} r_{K,A} + \sum_{r <_B v} \sum_{s <_A r} \text{kind}(s)(K)) \\
\Rightarrow & \sum_{K \in \mathcal{K}_{\text{vis}}} (v_{K,B} + \sum_{r <_B v} \text{kind}(r)(K)) = \sum_{K \in \mathcal{K}_{\text{vis}}} (v_{K,B} + \sum_{r <_B v} r_{K,A} + \sum_{s <_{B \circ A} v} \text{kind}(s)(K)) \\
\Rightarrow & \sum_{K \in \mathcal{K}_{\text{vis}}} (v_{K,B} + \sum_{r <_B v} \text{kind}(r)(K)) = \sum_{K \in \mathcal{K}_{\text{vis}}} (v_{K,B \circ A} + \sum_{s <_{B \circ A} v} \text{kind}(s)(K)) \\
& \langle B \text{ is sorted, } \text{kind}(v)(K) \neq \star, \text{ Lemma 3.14} \rangle \\
\Rightarrow & \text{cpc}_{\min}(v) \leq \sum_{K \in \mathcal{K}_{\text{vis}}} (v_{K,B} + \sum_{r <_B v} \text{kind}(r)(K)) = \sum_{K \in \mathcal{K}_{\text{vis}}} (v_{K,B \circ A} + \sum_{s <_{B \circ A} v} \text{kind}(s)(K))
\end{aligned}$$

KMM3 is similarly shown to hold. □

The sorting is also preserved by identities and tensor product. Therefore, it forms wide s-categories with kind interfaces as objects and kind sorted bigraphs as arrows.

**Definition 3.37** (kind sorted s-category/Brs with rigid min-max capacities). *The  $\Sigma$ -sorted (or kind sorted with rigid min-max capacities) bigraphs form a subcategory of  $\mathcal{BIG}(\mathcal{K}, \Phi)$  denoted by  $\mathcal{BIG}(\Sigma)$ . We call  $\mathcal{BIG}(\Sigma)$  a kind s-category with rigid min-max capacities. If  $\mathcal{R}$  is a set of  $\Sigma$ -sorted reaction rules then  $\mathcal{BIG}(\Sigma, \mathcal{R})$  is a  $\Sigma$ -sorted (or kind sorted Brs with rigid min-max capacities). The forgetful functor  $\mathcal{U} : \mathcal{BIG}(\Sigma) \rightarrow \mathcal{BIG}(\mathcal{K})$  is a sorting functor.*

Kind sortings with rigid capacities are a special case of the sorting above where  $kind_{\min}(K)(K') = kind_{\max}(K)(K')$  for all allowed pairs of controls  $K, K'$  (hence  $cpc_{\min}(K) = cpc_{\max}(K)$ ), and the set of interface sorts replaces  $\star$  with  $\odot$ .

### 3.6.1 Problems with kind sorting with capacities

Kind sortings with rigid min-max capacities are quite flexible; for example, we can specify in the signature that a  $K$ -node can contain at least one control and up to five consisting of one to three  $K$ -nodes, one to three  $L$ -nodes, and exactly one  $M$ -node. Alternatively, we can specify that a  $K$ -node will contain exactly one other node whose control is either  $K, L$ , or  $M$ . However, this extra flexibility negatively affects axiomatisations, specifically the definition of ions. We discuss ions in Section 4.2 but the basic problem is that each non-atomic control  $K$  will have multiple ions associated to it if  $kind_{\min}(K)(K') \neq kind_{\max}(K)(K')$  for any  $K'$ .

This problem would be avoided if we could allow interface sorts to have minimum and maximum capacities as well. However, adding minimum capacities breaks RPO creation at least in some cases; this can be demonstrated with a simple counterexample. Worse still, as we have added minimum capacities to controls, we cannot add maximum capacities to interface places as in the previous section. Adding maximum capacities to interface places guarantees that that capacity will not be exceeded but conditions KMM2 and KMM4 require that interface places guarantee that they will supply *at least* some number of nodes. Therefore, the sorting may not be preserved by composition. This could be solved by removing the kinds of interfaces from the right hand sides of the inequalities of KMM2 and KMM4 but this severely limits the bigraphs we can model; taking this approach, if  $kind_{\min}(K)(L) = n$  and  $G$  is a non-ground bigraph with a  $K$ -node then there must be at least  $n$   $L$ -nodes below the  $K$ -node. This would severely limit non-ground bigraphs<sup>13</sup>.

In short, we do not have a solution which allows both minimum and maximum capacities to be

<sup>13</sup>Although our thesis states that limitation is desirable!

added to bigraphs. We do not believe that there is a solution using the approach we have presented here.

## 3.7 Conclusions

In this chapter, we introduced kind sorting. The basic kind sorting allows us to structure the place graph using a containment relation between controls. We then extended this sorting by first adding the notion of invisible controls which allow controls some extra structure and then by adding exact, maximum, and minimum capacities to controls with some level of flexibility. Minimum capacities were not handled as well as maximum capacities due to problems involving minimal sorts which break RPO creation.

Applications of kind sortings include: i) modelling hierarchical grammars with bounded capacities and order amongst subterms; ii) adding simple type systems to bigraphical models of calculi; iii) modelling abstract systems whose containment relation is defined intuitively *e.g.* models of smart buildings or mobile commuters; iv) modelling simple semi-structured data like XML data or BibTeX entries.

We give examples of these applications in later chapters. In Chapter 11, we use a combination of kind sorting with semi-rigid maximum capacities with a sorting we define particularly for Milner's bigraphical model of the  $\lambda$ -calculus to present a Brs which models  $\alpha$ -equivalent classes of parallel  $\lambda$ -terms and almost only those terms. We then add simple types and intersection types to the kind sorting and show that the sorted bigraphs correspond to equivalence classes of type derivations. We are thus able to describe a Brs which can represent any terminating  $\lambda$ -term (and almost only such terms) and parallel compositions thereof. In Chapter 10, we present some examples of abstract systems which can be described using kind sorting as well as XML data and BibTeX entries.

Kind sorting also admits some interesting reactions to be described. By sorting the sites of bigraphs, we can encode some flow control and determinism into the reaction relation and even model simple algorithms. We expand on these ideas in Chapter 10. Note that we can guarantee that a site may *not* contain nodes of some control  $K$ ; we cannot guarantee that a site *will* contain a certain number of  $K$ -nodes *i.e.* we can specify *upper bounds* on how many of a particular control may be contained in a site but can only specify lower bounds on sites in the rigid sortings (the trivial case where the lower bound equals the upper bound). As explained in Section 3.6.1, we found it problematic to guarantee lower bounds and hence the existence of  $K$ -nodes under a root in the



non-rigid sortings. This begs the question<sup>14</sup> of whether this min-max sorting is *decomposable* [16].

Finally, the sorting allows us to better specify the place graph structure and the sorting of roots sites encodes some information about the exposed controls and allowed parameters respectively. The intensional nature of the sorting may marry well with the spatial logic BiLog. To motivate further work in this area, we discuss axiomatisations of the sortings in Chapter 4.

### 3.8 Further work

We have remarked that the sorting with rigid min-max capacities will have a more complicated axiomatisation so we will restrict our attention to certain signatures. In our presentation, the added generality introduces complications and, disappointingly, we cannot deal with minimum capacities as this breaks RPO creation. This leaves open the question of finding a sorting which has a simple axiomatisation and which can deal with both minimum and maximum capacities.

In pure bigraphs, controls are either active or passive so that they either admit all controls to react below them or else none. An *activity map* states at which sites a context is active. It may be possible to safely generalise this all-or-nothing activity for both the activity of controls and the notion of activity map as follows. First, generalise activity by defining  $actv : \mathcal{K} \rightarrow \mathcal{P}(\mathcal{K} \uplus \{site\})$ , assigning to each control the set of controls it allows be active below it. The extra element *site* specifies that a control allows sites below it to be active. We define all interface places to be active on  $\mathcal{K} \uplus \{site\}$ . We then say that a site  $s$  of a bigraph  $A$  is *active* if  $site \in actv(prnt_A(s))$  and  $ctrl(prnt^k(s)) \in actv(prnt^{k+1}(s)), k > 0$ . This may not suffice if it is required that activity is decomposable in which case it may be required to sort interface places. Combined with kind sorting, this generalisation of activity would allow us to model controls which can contain protected regions where activity is not allowed whilst allowing reaction in the other sublocations.

The remaining chapters in this part of the dissertation prove properties of kind sorted bigraphs.

In the next chapter, we present their static theory. We describe the epic and monic bigraphs, discuss axiomatisations for the different sortings, and define interesting subcategories which retain at least some of the pure theory such as congruence of bisimilarity in particular transition systems. These proofs of retention are given in Chapter 5 where we consider the properties of RPO creation and strong pushout reflection. We also consider weak pushout reflection which is easier to satisfy than the latter and has been shown by Bundgaard and Sassone to still retain some theory [24] and we introduce opcartesian pushout reflection which lies between these two and which we propose as

<sup>14</sup>Thanks to Lars Birkedal for this suggestion.

the best approximation to strong pushout reflection in kind sorted s-categories.

Finally, Chapter 6 introduces some other sortings we will use in the dissertation, simple operations to combine sortings, and ideas for alternative kind sortings where the ancestral tree is considered as opposed to the parent-child relationship.

# Chapter 4

## Static Theory

*If you don't have a date  
Celebrate  
Go out and sit on the lawn  
And do nothing*

*Waltz (better than fine) – Fiona Apple*

In this chapter we consider the static theory of kind sortings. We discuss the notions of *verticals*, special bigraphs which are identities besides an ‘inflation’ of the sorts of sites, and the related notions of opcartesian arrows and jointly opcartesian cospans. We then identify some useful subcategories of kind sortings which are later shown to retain some of the dynamic theory of pure bigraphs and which we will use in our models of typed  $\lambda$ -calculi.

### 4.1 Basic properties

We present the basic properties here in terms of the fundamental kind sorting. They generalise as expected to the more refined sortings. The missing proofs have been previously published [118] although not reviewed.

**Theorem 4.1** (kind bigraphs are wide monoidal). *Kind sorted  $s$ -categories are wide monoidal; the origin is  $\epsilon = \langle 0, \langle \rangle, \emptyset \rangle$ , and the interface  $\langle m, \vec{\theta}, X \rangle$  has width  $m$ .*

**Proposition 4.2** (isomorphisms in kind bigraphs). *A kind sorted bigraph  $\iota : I \rightarrow J$  is an isomorphism iff it has no nodes, its parent map and link maps are bijections and the place graph bijection respects kind meaning that if  $\iota(s) = r$  then  $\text{kind}(s) = \text{kind}(r)$ . The isomorphisms are combinations of a kind sorted place graph isomorphism and a link graph isomorphism.*

$\mathcal{U}$  preserves isomorphisms but does not reflect them [118].

Unlike isomorphisms, epis and monos remain unchanged from the pure place graph theory.

**Proposition 4.3** (epis and monos). *A kind sorted bigraph is epi (resp. mono) iff its underlying pure bigraph is epi (resp. mono).*

**Corollary 4.4.**  *$\mathcal{U}$  both preserves and reflects epis and monos.*

Note that if a kind place graph is both an epimorphism and a monomorphism then it may not be an isomorphism. Thus, kind s-categories are not *balanced* categories.

A kind sorted bigraph  $G$  may always be decomposed as  $J \uparrow I \circ F$  where  $F$  has the least outer sorting which satisfies the sorting conditions and where  $J \uparrow I$  is a vertical, an identity on places and links which ‘inflates’ the outer sorting of  $F$  to match that of  $G$ . Similarly, for any cospan  $G_i : H_i \rightarrow J, i \in \{0, 1\}$  in a kind sorted s-category, there always exists a pair  $F_i : H_i \rightarrow I$  such that  $I$  has a least sort for the pair  $i$  and  $G_i = J \uparrow I \circ F_i$ . These properties can be succinctly described by saying that arrows (resp. cospans) of kind sortings are nearly opcartesian (resp. nearly jointly opcartesian). The sort of the interfaces  $I$  in the examples can be given by simple constructions.

**Definition 4.5** (least sort for an arrow). *The least sort  $\theta$  of an arrow  $G$  of a fundamental kind sorting is defined pointwise on the roots  $r$  of the arrow by:*

$$\theta = \bigcup_{v <_G r} \text{ctrl}(v) \cup \bigcup_{s <_G r} \text{kind}(s).$$

**Definition 4.6** (least sort for a cospan). *The least sort  $\theta$  of a cospan  $\vec{G}$  of a fundamental kind sorting is defined pointwise on the places  $r$  of the shared interface by:*

$$\theta = \bigcup_{v <_{G_i} r, i \in \{0, 1\}} \text{ctrl}(v) \cup \bigcup_{s <_{G_i} r, i \in \{0, 1\}} \text{kind}(s).$$

**Definition 4.7** (partial order of kind sorts). *The partial order  $\preceq$  on control and interface sorts for a fundamental kind sorting is defined as subset inclusion.*

**Terminology** (inflation). *An inflation is defined as a vertical in a kind sorted s-category.*

Inflations are used in our model of intersection typed  $\lambda$ -calculus to capture a typing rule which resolves a sort of finite polymorphism into a more basic type (the intersection elimination rule).

### 4.1.1 Hard bigraphs

**Definition 4.8** (hard kind bigraphs). *A hard kind bigraph is one in which no root or non-atomic node is barren. They form a sub-s-category denoted by  $\mathcal{BIG}_h(\mathcal{K})$ .*

Hard kind bigraphs are epi by definition. As in the pure theory, if a composition  $B \circ A$  in  $\mathcal{BIG}(\Sigma_K)$  is hard then so are both  $A$  and  $B$ .  $\mathcal{U}$  maps hard kind bigraphs to hard bigraphs.

**Proposition 4.9** (pushouts for hard bigraphs). *If  $\vec{A}$  is a consistent pair<sup>1</sup> of hard kind sorted bigraphs, then it has a pushout  $\vec{B}$  which is hard and is a pushout in  $\mathcal{BIG}_h(\Sigma_K)$ .*

We now highlight another connection between  $\mathcal{BIG}(\Sigma_K)$  and  $\mathcal{BIG}_h(\Sigma_K)$ , generalising the similar connection of pure bigraphs. Let  $\mathcal{K}$  be a kind signature. We define a new atomic control  $\Delta$  with zero arity; we adjoin  $\Delta$  to  $\mathcal{K}$  to form  $\mathcal{K}^\Delta = \mathcal{K} \uplus \Delta$  and define  $kind_{\mathcal{K}^\Delta}$  by

$$kind_{\mathcal{K}^\Delta}(K) = \begin{cases} \emptyset & \text{if } kind_{\mathcal{K}}(K) = \emptyset \\ kind_{\mathcal{K}}(K) \cup \Delta & \text{else} \end{cases}$$

so that all non-atomic controls of  $\mathcal{K}$  can contain  $\Delta$ . We can make any arrow  $G$  of  $\mathcal{BIG}(\mathcal{K})$  with signature  $\mathcal{K}$  into a hard kind bigraph of  $\mathcal{BIG}_h(\mathcal{K}^\Delta)$  with signature  $\mathcal{K}^\Delta$  by adding a  $\Delta$ -node as a child of any barren root or non-atomic node, adding  $\Delta$  to the sort of any barren root.

Two bigraphs are said to be *place-equivalent* if they are equal up to occurrences of  $\Delta$ -nodes. Place equivalence,  $\equiv_\Delta$ , for kind bigraphs is defined as in the pure theory and is a static congruence<sup>2</sup>. There is also a quotient s-category  $\mathcal{BIG}_h(\mathcal{K}^\Delta)/\equiv_\Delta$ . Unlike the pure theory,  $\mathcal{BIG}_h(\mathcal{K}^\Delta)/\equiv_\Delta$  is not isomorphic with  $\mathcal{BIG}(\mathcal{K})$ . This can be seen by noting that the objects of  $\mathcal{BIG}(\mathcal{K})$  are a strict subset of the objects of  $\mathcal{BIG}_h(\mathcal{K}^\Delta)/\equiv_\Delta$ , as the interfaces of the latter s-category may contain  $\Delta$  in their sorts. This can be dealt with by considering the largest full subcategory where all interface sorts contain  $\Delta$ .

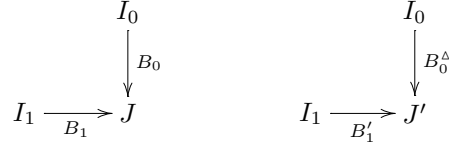
In the pure theory, the operation of adding a new place node  $\Delta$  to a place graph was easily defined. Since we have to ensure that our place graphs obey the kind rules, we must define the operation of adding a new place node  $\Delta$  to a kind bigraph more carefully. Given a kind bigraph  $G : I \rightarrow \langle n, \vec{\theta}_J \rangle$ , if we add a  $\Delta$ -node as a child of any non-atomic node, then we have a new bigraph  $G' : I \rightarrow \langle n, \vec{\theta}_{J'} \rangle$  in the same homset as  $G$ . If we add a  $\Delta$ -node as a child of any root  $r \in n$  in  $G$ , then we have a new bigraph  $G' : I \rightarrow \langle n, \vec{\theta}_{J'} \rangle$  which may have a different outer interface to  $G$ . We define

$$\vec{\theta}_{J'} \stackrel{\text{def}}{=} \langle \theta_0 \dots \theta_r \cup \Delta \dots \theta_{n-1} \rangle.$$

<sup>1</sup>See Appendix A.1 and Proposition 5.31, page 97 for the consistency conditions for kind bigraphs.

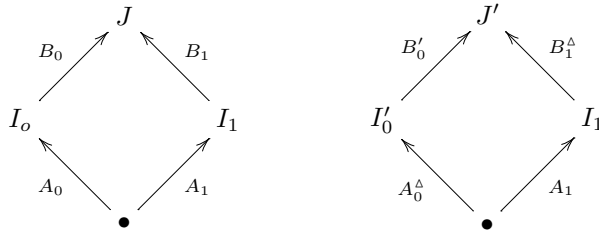
<sup>2</sup>If  $f \equiv_\Delta f'$  and  $g \equiv_\Delta g'$  then (when both sides of the equations are defined),  $f \otimes g \equiv_\Delta f' \otimes g'$  and  $f \circ g \equiv_\Delta f' \circ g'$ .

With this definition, given a least sorted interface  $J$  for a pair  $\vec{B} : \vec{I} \rightarrow J$ , if we add a place node to  $B_0$  to yield the pair  $(B_0^\Delta, B'_1)$  as in the figure below, with outer interface  $J'$ , and where  $\text{prnt}_{B'_1} = \text{prnt}_{B_1}$ , then  $J'$  is a least sorted interface for  $B_0^\Delta, B'_1$ .  $J'$  has a least sorting as either  $J' = J$  or else  $J$  has been altered as above such that K1 is satisfied in  $B_0^\Delta$ .



In the following final propositions of this section, we shall use  $\Delta$  to mean a fresh place node  $\Delta_u$ , of control  $\Delta$ , distinct from all others present. The final two propositions in this section are not exactly the same as in the pure theory. The extra complications lie in the fact that a bound must be jointly opcartesian in order to be an IPO for some pair. Note, however, that if all the bigraphs in the propositions are opcartesian (as in the rigid sortings and particular sub-categories defined later) then the extra complication does not arise and we retrieve the original propositions exactly. In the following,  $\text{BIG}(\Sigma_K^\Delta)$  denotes the kind sorted category over  $\mathcal{K}^\Delta$ .

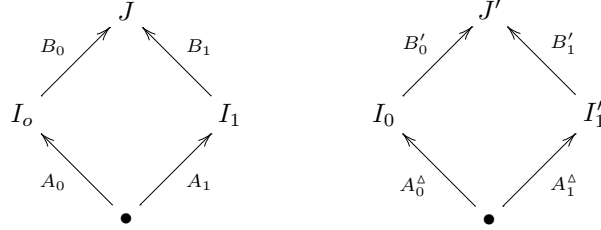
**Proposition 4.10** (first pushout variation). *Let  $\vec{B} : \vec{I} \rightarrow J$  be a bound for  $\vec{A}$  in  $\text{BIG}_h(\Sigma_K^\Delta)$  where  $\vec{\theta}$  is the vector of sorts of  $J$ . Add a new place node  $\Delta$  to both  $A_0$  and  $B_1$ , yielding  $A_0^\Delta$  and  $B_1^\Delta$  such that (i) if  $\Delta$  is added as a child of root  $r$  in  $B_1$  and  $\Delta \in \theta_r$ , then  $\Delta \in \theta_r$  is necessary for K1 or K2 to hold in  $\vec{B}$  and (ii)  $B'_0 \circ A_0^\Delta = B_1^\Delta \circ A_1$ . Then  $\vec{B}$  is a pushout for  $\vec{A}$  iff  $(B'_0, B_1^\Delta)$  is a pushout for  $(A_0^\Delta, A_1)$ .*



Again, the following proposition needs to be altered due to the kind rules. This results in the same kind of asymmetry with regards the fitness of  $J$  and  $J'$  to  $\vec{B}$  and  $\vec{B}'$  respectively.

**Proposition 4.11** (second pushout variation). *Let  $\vec{B}$  be a bound for  $\vec{A}$  in  $\text{BIG}_h(\Sigma_K^\Delta)$ . Let a fresh place node  $\Delta$  be added to both members of  $\vec{A}$ , yielding  $\vec{A}^\Delta$  and  $\vec{B}'$  (where  $\text{prnt}_{B'_i} = \text{prnt}_{B_i}$  and  $J'$  is equal to  $J$  except for any change needed so that K2 holds in  $B'_1$ ) such that  $\vec{B}'$  is also a bound for  $\vec{A}^\Delta$ , and with  $A_0^\Delta(\Delta)$  a node (not a root). Also, if  $\Delta$  is added as a child of root  $s$  in  $A_1$ ,  $B_1(s) = r$  and  $\Delta \in \theta_{J,r}$ , then  $\Delta \in \theta_{J,r}$  is necessary for K2 to hold in  $B_1$ . Then*

1. If  $\vec{B}$  is a pushout for  $\vec{A}$ ,  $\vec{B}'$  is a pushout for  $\vec{A}^\Delta$ .
2. Let  $\Delta$  have a sibling  $w$  in both  $A_0^\Delta$  and  $A_1^\Delta$ . Then if  $\vec{B}'$  is a pushout for  $\vec{A}^\Delta$ ,  $\vec{B}$  is a pushout for  $\vec{A}$ .



Finally, given that hard kind bigraphs have no barren roots or non-atomic nodes, we may also consider altering a non-hard kind bigraph to be hard by assigning a site  $s$ , where  $\theta_s = \emptyset$ , as a child to each barren root or non-atomic node. Such a site is essentially useless as no node can ever be planted inside it. This direction has not been pursued to date.

## 4.2 Further properties

The special IPOs propositions [74, Propositions 9.8, 9.9, 9.11] hold for fundamental kind sortings. We also present a new IPO proposition, Proposition 4.13.

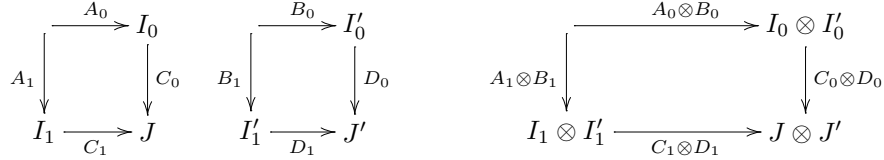
**Proposition 4.12** (containment pushout). *Let  $A$  be epi. Then the pair  $(A, F \circ A)$  has the pair  $(F, \text{id})$  as a pushout. In particular, by taking  $A = \text{id}$  and  $F = \text{id}$  respectively: (1) any pair  $(\text{id}, F)$  has  $(F, \text{id})$  as a pushout, and (2) if  $A$  is epi then  $(A, A)$  has  $(\text{id}, \text{id})$  as a pushout.*

It seems that any pair  $(J \uparrow I \circ R, R)$  will have a unique IPO (pushout)  $(C_0, C_1)$ , where  $C_0$  and  $C_1$  have no nodes and injective link and parent maps. However, we are especially interested in the case where  $R$  is epi since it is usually required that redexes in a Brs are epi. We present a related proposition below.

**Proposition 4.13** (inflation IPO). *Let  $R : I \rightarrow J$  be epi. Then the pair  $(J' \uparrow J \circ R, R)$  has the pair  $(\text{id}, J' \uparrow J)$  as a pushout.*

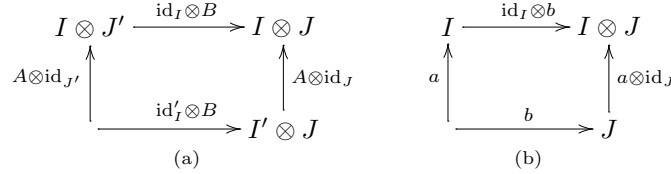
Proposition 4.13 may be useful when discussing the standard transition system (see Definition 5.5, page 91). For example, let  $a = (J \uparrow I \circ r)$  for some ground redex  $r$  i.e.  $a$  is an ‘inflated redex’. Then there is a unique transition in ST with respect to that redex whose label is  $\text{id}$ .

**Proposition 4.14** (tensor IPO ). *In  $s$ -categories of (hard) kind place graphs, link graphs, or (hard) kind bigraphs, let  $\vec{C}$  be an IPO for  $\vec{A}$  and  $\vec{D}$  be an IPO for  $\vec{B}$ , where the supports of the two IPOs are disjoint. Then, provided the tensor products exist,  $\vec{C} \otimes \vec{D}$  is an IPO for  $\vec{A} \otimes \vec{B}$ .*



**Corollary 4.15** (tensor IPOs with identities ). *Let  $A : I' \rightarrow I$  and  $B : J' \rightarrow J$  share no nodes, and let the names of  $I', I$  be disjoint from those of  $J', J$ . Then the pair  $(A \otimes \text{id}_{J'}, \text{id}_{I'} \otimes B)$  has an IPO  $(\text{id}_I \otimes B, A \otimes \text{id}_J)$ . See diagram (a).*

*In particular if  $I' = J' = \epsilon$  then  $A = a$  and  $B = b$  are ground bigraphs, and the IPO as in diagram (b).*



As in the pure theory, we shall: (1) call a kind bigraph *lean* if its link graph is lean; and (2) denote the result of adding a set  $E$  of fresh, idle edges to a kind bigraph  $A$  as  $A^E$ .

For the following proofs, note that  $\mathcal{U}$  preserves and reflects leanness *i.e.*  $A$  is lean iff  $A^u$  is lean, as their link graphs are equal. Note also that  $A$  and  $A^E$  are in the same homset – in particular, their outer interfaces are equal – and that adding or removing idle edges to a consistent pair of bigraphs (pure or kind) yields a consistent pair of bigraphs.

**Proposition 4.16** (IPOs, idle edges and leanness ). *For any two pairs  $\vec{A}$  and  $\vec{B}$ :*

1. *If  $\vec{B}$  is an IPO for  $\vec{A}$ , and  $A_1$  is lean, then  $B_0$  is lean.*
2. *For any fresh set  $E$  of edges,  $\vec{B}$  is an IPO for  $\vec{A}$  iff  $(B_0, B_1^E)$  is an IPO for  $(A_0^E, A_1)$ .*

Lean-support equivalence for kind bigraphs remains unchanged from the pure theory *i.e.* “two concrete (kind) bigraphs  $A$  and  $B$  are lean-support equivalent, written  $A \approx B$ , if after discarding any idle edges they are support equivalent” [74]. Lean-support equivalence is a static congruence and so we have quotient functors to *abstract bigraphs* (Definition A.3, Appendix A.1) as follows.



**abstract kind bigraphs** An abstract kind bigraph is a lean-support equivalence class of concrete kind bigraphs. We can define the category  $\text{BIG}(\Sigma_K)$  (resp.  $\text{BIG}_h(\Sigma_K)$ ) of *abstract kind (hard) bigraphs* as having the same objects as  $\mathcal{B}\text{IG}(\Sigma_K)$  (resp.  $\mathcal{B}\text{IG}_h(\Sigma_K)$ ), where its arrows are lean-support equivalence classes of concrete kind bigraphs. For any signature  $\mathcal{K}$ , we have quotient functors  $\llbracket \cdot \rrbracket : \mathcal{B}\text{IG}(\Sigma_K) \rightarrow \text{BIG}(\Sigma_K)$  and  $\llbracket \cdot \rrbracket : \mathcal{B}\text{IG}_h(\Sigma_K) \rightarrow \text{BIG}_h(\Sigma_K)$  sending a concrete (hard) kind bigraph to its lean-support equivalence class.

**ground bigraphs** A *ground* bigraph is one with inner face  $\epsilon = \langle 0, \langle \rangle, \emptyset \rangle$ .

**prime interfaces, bigraphs** An interface  $I = \langle m, \vec{\theta}, X \rangle$  is prime if it has width  $m = 1$ . A prime kind bigraph has no inner names and a prime outer face.

**merge** The definition of the prime  $\text{merge}_m$  is modified to exactly satisfy K2. It is defined as  $\text{merge}_m : \langle m, \vec{\theta}, \emptyset \rangle \rightarrow \langle 1, \langle \theta \rangle, \emptyset \rangle$  where  $\theta = \bigcup_{i \in m} \theta_i$ .  $\text{merge}_m$  has no nodes and maps  $m$  sites to a single root which can contain the union of the controls that its children can contain.  $\text{merge}_m$  is opcartesian.

**wirings and discreteness** Wirings and discreteness are as before and pertain only to link graphs.

**ions, atoms, molecules** We define an ion as in the pure theory except that the site can contain all controls that the node can. For any non-atomic control  $K$  with arity  $k$  and sequence  $\vec{x}$  of  $k$  distinct names, we define the *discrete ion*  $K_{v, \vec{x}} : \langle 1, \langle \text{kind}(K) \rangle, \emptyset \rangle \rightarrow \langle 1, \langle \{K\} \rangle, \vec{x} \rangle$  to have a single  $K$ -node  $v$ , whose ports are severally linked to  $\vec{x}$ . The site is a child of the node. For atomic  $K$ , a *discrete atom* is  $K_{v, \vec{x}} : \epsilon \rightarrow \langle 1, \langle \{K\} \rangle, \vec{x} \rangle$ , again containing a single  $K$ -node  $v$  whose ports are severally linked to  $\vec{x}$ .

For any prime discrete  $P$  with outer face  $\langle 1, \langle \theta \rangle, Y \rangle = \langle 1, \langle \text{kind}(K) \rangle, Y \rangle$  we call  $(K_{v, \vec{x}} \otimes \text{id}_Y) \circ P$  a *discrete molecule*.

Ions, atoms, and molecules are defined to be discrete and opcartesian. Arbitrary (non-discrete, nearly opcartesian) ions, molecules and atoms are constructed by precomposing by  $\omega \otimes J \uparrow I$ , where  $\omega$  is a wiring, with a discrete ion, molecule, or atom.

Our final section of the static theory for kind bigraphs defines some operations and decompositions of pure bigraphs for kind bigraphs.

### 4.2.1 Operations and decompositions

**parallel product** We extend the definition of parallel product for pure bigraphs (Appendix A.1, Definition A.4). The *parallel product* of two interfaces is defined as

$$\langle m, \vec{\theta}_0, X \rangle \| \langle n, \vec{\theta}_1, Y \rangle \stackrel{\text{def}}{=} \langle m + n, \vec{\theta}_0 \vec{\theta}_1, X \cup Y \rangle.$$

The parallel product of two bigraphs is identical to the pure definition. We repeat it here:

$$G_0 \| G_1 \stackrel{\text{def}}{=} \langle G_0^P \otimes G_1^P, G_0^L | G_1^L \rangle : I_0 \otimes I_1 \rightarrow J_0 \| J_1$$

where the place graphs are kind-sorted, the interfaces exist, and the node sets are disjoint. There is an alternative definition of parallel product as in the pure theory [74, Proposition 9.14] (see Appendix A.1, Definition A.5).

The parallel product operation remains associative (we need only check that  $(\vec{\theta}_0 \vec{\theta}_1) \vec{\theta}_2 = \vec{\theta}_0 (\vec{\theta}_1 \vec{\theta}_2)$ ), with unit  $\epsilon$ . Note that, as for pure bigraphs, the parallel product requires that node sets are disjoint but does not require that the edge sets are disjoint. This is because the *link* map of  $G_0^L | G_1^L$  is defined as the union of the constituent link maps.

The parallel product of two opcartesian bigraphs is an opcartesian bigraph.

**prime product** The *prime product* of two interfaces is defined as

$$\langle m_0, \vec{\theta}, X_0 \rangle | \langle m_1, \vec{\theta}', X_1 \rangle \stackrel{\text{def}}{=} \langle 1, \bigcup_{i \in m_0} \theta_i \cup \bigcup_{i \in m_1} \theta'_i, X_0 \cup X_1 \rangle.$$

For two prime bigraphs  $\vec{P} : \vec{I} \rightarrow \vec{J}$  with disjoint support, if  $P_0 \| P_1$  is defined and  $m$  is the sum of the widths of  $J_0$  and  $J_1$ , the *prime product* is defined, as in pure bigraphs, as

$$P_0 | P_1 \stackrel{\text{def}}{=} \text{merge}_m \circ (P_0 \| P_1) : I_0 \otimes I_1 \rightarrow J_0 | J_1.$$

The prime product operation remains associative, with unit  $\langle 1, \langle \emptyset \rangle, \emptyset \rangle$  when applied to primes. The prime product of two opcartesian bigraphs  $P_0, P_1$  is an opcartesian bigraph. This can also be seen by observing that both  $P_0 \| P_1$  and  $\text{merge}_m$  are opcartesian.

**underlying discrete bigraph** The factorisation of a bigraph  $G$  to a discrete normal form presented in the pure theory (Appendix A.1, Definition A.7) remains unchanged as it pertains only to the link graph.

It may be necessary, perhaps for an axiomatisation of kind bigraphs, to define something akin to discreteness for kind bigraphs. This notion follows.

**Proposition 4.17** (underlying discrete, opcartesian (kind sorted) bigraph). *Every kind bigraph  $G : H \rightarrow J$  in  $\mathcal{BIG}(\Sigma_K)$  can be expressed uniquely (up to iso) as  $G = (\omega \otimes J \uparrow I) \circ D$ , where  $\omega$  is a wiring,  $D$  is discrete and opcartesian, and  $J \uparrow I$  has no names<sup>3</sup>.*

Composition and tensor product preserve discreteness, as for pure bigraphs, and thus in  $\mathcal{BIG}(\Sigma_K)$  and  $\mathcal{BIG}_h(\Sigma_K)$ , the discrete bigraphs form a monoidal sub-s-category. Likewise, the composition of two opcartesian bigraphs is an opcartesian bigraph and the tensor product of two opcartesian bigraphs is an opcartesian bigraph. Thus we have that in  $\mathcal{BIG}(\Sigma_K)$  and  $\mathcal{BIG}_h(\Sigma_K)$ , the discrete, opcartesian bigraphs form a sub-s-category of *fitting* bigraphs (see the next section) as  $\gamma_{I,J}$  is a discrete, opcartesian bigraph.

The next proposition slightly extends the original [74] to give another factorisation of a discrete bigraph which includes opcartesians. Parts 1, 3 and 4 of the proposition are as in the pure theory (omitting the vectors  $\vec{\theta}$ ) and so are not reproduced in the appendices. For the following proofs, note that  $A$  is discrete iff  $A^u$  is discrete as they have the same link graph.

**Proposition 4.18** (synthesis and analysis of discrete bigraphs). *In  $\mathcal{BIG}(\Sigma_K)$  or  $\mathcal{BIG}_h(\Sigma_K)$  the discrete kind bigraphs form a monoidal sub-s-category. Moreover*

1. *Every discrete  $D : \langle m, \vec{\theta}_I, X \rangle \rightarrow \langle n, \vec{\theta}_J, Y \rangle$  may be factored uniquely, up to isomorphism on the domain of each factor  $D_i$ , as*

$$D = \alpha \otimes ((D_0 \otimes \cdots \otimes D_{n-1}) \circ \pi)$$

*with  $\alpha$  a renaming, each  $D_i$  prime and discrete, and  $\pi$  a permutation.*

2. *Every discrete  $D : \langle m, \vec{\theta}_I, X \rangle \rightarrow \langle n, \vec{\theta}_J, Y \rangle$  may be factored uniquely, up to isomorphism on the domain of each factor  $D_i$ , as*

$$D = \alpha \otimes (i \circ (D_0 \otimes \cdots \otimes D_{n-1}) \circ \pi)$$

*with  $\alpha$  a renaming,  $i$  an inflation, each  $D_i$  prime, discrete and opcartesian, and  $\pi$  a permutation.*

3. *If  $(D', G')$  is an IPO for  $(G, D)$  and  $D$  is discrete, then  $D'$  is discrete.*
4. *If  $D' \circ G = \omega D$  with  $D$  and  $D'$  discrete, then  $(D', \omega)$  is an IPO for  $(G, D)$ .*

The remainder of this section is self-contained and any propositions are copied directly from the pure theory to which the reader is encouraged to refer to for more details.

<sup>3</sup>In the equation  $G = (\omega \otimes J \uparrow I) \circ D$ ,  $D$  can be described as ‘taking  $G$  and pulling the wiring to the outer interface’ and  $(\omega \otimes J \uparrow I)$  can be described as ‘rejoining these wires as open or closed links’.

**instantiation** An instantiation  $\varrho$  from  $m$  to  $n$ , which is written  $\varrho :: m \rightarrow n$ , is determined by a function  $\bar{\varrho} : n \rightarrow m$ . For any  $X$  this function defines the map

$$\varrho : \text{Gr}\langle m, (\theta_0 \cdots \theta_{m-1}), X \rangle \rightarrow \text{Gr}\langle n, (\theta_{\bar{\varrho}(0)} \cdots \theta_{\bar{\varrho}(n-1)}), X \rangle$$

as follows. Decompose  $g : \epsilon \rightarrow \langle m, (\theta_0 \cdots \theta_m), X \rangle$  into  $g = \omega(d_0 \otimes \cdots \otimes d_{m-1})$ , with  $Y = \bigsqcup_{i \in m} Y_i$ ,  $\omega : Y \rightarrow X$  and each  $d_i : \epsilon \rightarrow \langle 1, (\theta_i), Y_i \rangle$  prime and discrete. Then define

$$\varrho(g) \stackrel{\text{def}}{=} \omega(e_0 || \cdots || e_{n-1}),$$

where  $e_j \simeq d_{\bar{\varrho}(j)}$  for  $j \in n$ . If the map  $\varrho$  is injective then the instantiation  $\bar{\varrho}$  is said to be *affine*.

An instantiation which is affine does not replicate any factors of the parameter  $d$ . Instantiations come into play in the dynamic theory of bigraphs and so will not be discussed here. However, it is worth noting that Brss whose reaction rules are simple, prime, and affine have nice properties [113].

Recall that support equivalence is defined on homsets and so if  $G \simeq G'$  then the inner and outer  $\vec{\theta}$  vectors of both bigraphs are respectively identical. Note also that since each  $d_i$  is discrete, it has no edges and so the  $e_j$  in the definition also have no edges. Support equivalence here is just a renaming of the node set of  $d_i$ . This also means that in the bigraph  $\varrho(g)$ , multiple copies of a bigraph  $d_i$  will share the same links.

**wiring an instance** The proof that wiring commutes with instantiation *i.e.*  $\omega\varrho(a) \simeq \varrho(\omega a)$ , stands.

### 4.3 Subcategories of kind sortings

A place sorting is a subcategory of an s-category with sorted interfaces where the bigraphs satisfy certain conditions. In this section, we study subcategories of kind sortings. Our motivation is threefold.

Most importantly, subcategories allow a closer modelling of some systems or calculi. For example, we can describe the sorting Milner used to model finite CCS as a particular subcategory of a kind sorting with disjoint sorts, we use a more general type of subcategory to model simply typed  $\Lambda_{\text{sub}}$ , and we use a subcategory with intersections of sorts to model intersection typed  $\Lambda_{\text{sub}}$ . Forming subcategories also: i) may remove bigraphs which do not correspond to terms of the calculus; ii) may reduce the set of sorts allowed in interfaces, allowing a static correspondence between place sorts and types of typed calculi (see Chapter 11); and iii) reduce the set of labels

in some of the transition systems, allowing us to demonstrate an operational correspondence with calculi and their bigraphical modelling.

Secondly, subcategories may have stronger properties. For example, kind sortings do not strongly reflect pushouts in general but certain subcategories do. This leads us to study subcategories where arrows have stronger properties. Conversely, certain subcategories can not guarantee RPO creation. Gathering intuitions or proofs as to when subcategories satisfy these properties helps when defining new sortings.

Finally, much of the literature studies sorting functors which has led to some general theorems of safety [92, 113, 73, 16]. Subcategories are not surjective on objects in general and so do not fit the definition of sorting functor. Our contribution here is to begin that study in some detail for kind sortings. Hopefully some of our closing observations will generalise to other sortings.

**Definition 4.19** (interface sorts). *The (set of) interface sorts of a subcategory of a kind  $s$ -category  $\mathcal{BIG}(\Sigma_K)$  is the smallest set which contains the sort of any place of any interface in the  $s$ -category, defined by*

$$\{\theta \mid \theta \in \vec{\theta}, \langle m, \vec{\theta}, X \rangle \in \text{obj}(\mathcal{BIG}(\Sigma_K))\}.$$

Conversely, we can also define full subcategories by specifying the set of interface sorts which places of interfaces are allowed to have. This is the approach we take here. For other types of subcategories, we require certain properties of arrows with respect to the functor.

**Definition 4.20** (control sorts). *Let  $\mathcal{K}'$  denote the subset of non-atomic controls of a signature  $\mathcal{K}$ . The (set of) control sorts of an  $s$ -category over  $a$ :*

- *fundamental kind signature, kind signature with visibility, or kind signature with rigid capacities is defined as  $\bigcup_{K \in \mathcal{K}'} \text{kind}(K)$ ;*
- *kind signature with (rigid/semi-rigid) maximum capacities is defined as*

$$\bigcup_{K \in \mathcal{K}'} (\text{cpc}(K), \text{kind}(K));$$

- *kind signature with rigid min-max capacities is defined as*

$$\bigcup_{K \in \mathcal{K}'} (\text{kind}_{\min}(K), \text{cpc}_{\min}(K), \text{cpc}_{\max}(K), \text{kind}_{\max}(K)).$$

A subcategory  $\mathcal{I} : \mathcal{C} \hookrightarrow \mathcal{D}$  of a sorting  $\mathcal{U} : \mathcal{D} \hookrightarrow \mathcal{E}$  is not necessarily a sorting functor itself. In most of the examples we consider, the composition  $\mathcal{UI}$  will be but for some cases, the following terminology will help disambiguate.

**Definition 4.21** (pure functor of a subcategory, underlying pure interface/bigraph). *Let  $\mathcal{I} : \mathcal{C} \hookrightarrow \mathcal{D}$  be a subcategory of a sorting  $\mathcal{U} : \mathcal{D} \hookrightarrow \mathcal{E}$  where  $\mathcal{E}$  is an s-category of bigraphs with a pure place graph. The composition  $\mathcal{UI}$  is called the pure functor of the subcategory. For any interface  $J$  (resp. bigraph  $G$ ) in  $\mathcal{C}$ , we call  $\mathcal{UI}(J) = J^u$  (resp.  $\mathcal{UI}(G) = G^u$ ) the underlying pure interface (resp. bigraph).*

We concentrate our investigation on subcategories of kind sortings with visibility<sup>4</sup>. Our reason for this is that the interface sorts of the sorted s-categories are a powerset and have a full lattice structure. These makes it simple to define various sets of interface sorts and reason about them. When we move to the more general sortings, we must worry about the extra dimensions concerning capacity as in the definition of control sorts. We leave this to future work. Note that the control sorts of an s-category over a kind signature with visibility consist only of visible controls.

We begin by defining a type of subcategory where every bigraph has a least sort in some sense.

**Definition 4.22** (fitting subcategory). *A fitting subcategory  $\mathcal{I} : \mathcal{C} \rightarrow \mathcal{BIG}(\Sigma_K)$  is a subcategory of a kind sorting  $\mathcal{U} : \mathcal{BIG}(\Sigma_K) \rightarrow \mathcal{BIG}(K)$  with the following properties. Let  $G$  be a bigraph of  $\mathcal{C}$  with outer interface  $I$ . Then  $I$  is the smallest interface of  $\mathcal{BIG}(\Sigma_K)$  such that  $\mathcal{I}(G)$  is sorted where  $\theta$  is ‘smaller than’  $\theta'$  if  $\theta \subset \theta'$  or  $|\theta| < |\theta'|$ .*

The key properties of fitting s-categories is that we always choose the ‘smallest’ outer interface and that a smallest interface exists. In particular, no two distinct sorts of the same cardinality may be used to sort a place of the outer interface of the same bigraph. This means that two bigraphs in the fitting s-category with identical parent maps, link graphs, inner interfaces, and outer names must be equivalent.

**Definition 4.23** (some full and fitting subcategories). *A full or fitting subcategory is defined by defining the set  $Int$  of interface sorts. The interface sorts must be a subset of  $\mathcal{P}(K_{vis})$ . In the following, we do not require that  $\emptyset$  is an interface sort. Let  $Ctrl$  denote the set of control sorts. A full or fitting subcategory is:*

- a bounded meets s-category<sup>5</sup> if, given any non-empty set  $S$  of interface sorts which has a lower bound in  $(Int \cup Ctrl, \subseteq)$  which is not  $\emptyset$ , there exists a meet for  $S$  in  $(Int, \subseteq)$ .
- a meet s-category if  $Int$  is closed under intersection with itself and with  $Ctrl$  i.e. for each pair  $\theta, \theta'$  where  $\theta$  is an interface sort and  $\theta'$  is an interface sort or control sort,  $\theta \cap \theta'$  is an interface sort;

<sup>4</sup>The reasoning should also hold for fundamental kind sortings.

<sup>5</sup>Following the terminology of Hennessy and Riely [68].

- downward closed if  $Int$  is a downward closed set of  $(\mathcal{K}_{vis}, \subseteq)$  i.e. for each interface sort  $\theta$ , any subset of  $\theta$  besides  $\emptyset$  is an interface sort;
- controlled if  $Ctrl \subseteq Int$  i.e. the set of interface sorts contains the set of control sorts;
- bounded complete if, given any non-empty set  $S$  of interface sorts which has an upper bound in  $(Int \cup Ctrl, \subseteq)$ , there exists a join for  $S$  in  $(Int, \subseteq)$ ;
- unioned if  $Int$  is closed under union;
- opcartesian if the subcategory is fitting and the set of interfaces sorts is  $\mathcal{P}(\mathcal{K}_{vis})$ ;
- partitioned if distinct interface sorts are pairwise disjoint;
- fully partitioned if it is partitioned and each  $K \in \mathcal{K}_{vis}$  is an element of some interface sort;
- homomorphic if it is sorted with a homomorphic kind sorting and the interface sorts are exactly the homomorphic groupings (see Definition A.9).

We consider partitioned, fully partitioned, and homomorphic subcategories to be both fitting and full so we omit these qualifiers for brevity.

These definitions identify more types of fitting s-categories than in previous work [118, 119]. The notion we previously called ‘fitting bigraphs’ is now a particular case of the above definition, the opcartesian subcategory. This fitting subcategory only removes bigraphs; in general, the other fitting subcategories remove interfaces as well.

There are three flavours of subcategory here.

The opcartesian subcategory is a particular downward closed subcategory<sup>6</sup> which are examples of meet s-categories which are themselves examples of bounded meets subcategories. These s-categories satisfy the property that lower bounds of sets of interface sorts always exist. We only consider meet s-categories here but as we will see that bounded complete subcategories are important for proofs of RPO creation, bounded meets subcategories may suffice for proofs that jointly opcartesian pairs reflect pushouts.

A unioned subcategory is a special type of bounded complete subcategory.

A homomorphic s-category is a special case of a fully partitioned subcategory which is a special case of a partitioned subcategory. These subcategories satisfy the property that intersections or

---

<sup>6</sup>It is also a unioned subcategory, bounded complete, and controlled subcategory.

unions of pairs of interface sorts only exist when both sorts are equal (or unless  $\emptyset$  is an interface sort). An important difference between general partitioned subcategories and homomorphic subcategories is that the latter also implies restrictions on the sorting of *controls*.

In previous work, we showed that both the opcartesian subcategory and any homomorphic subcategory reflect pushouts<sup>7</sup>. These results are generalised in Chapter 5. Table 5.2 summarises the properties of the various subcategories of an arbitrary kind s-category. A negative mark means that in the general case, this property is not guaranteed. Counterexamples for RPO creation of meet and downward closed s-categories and for pushout reflection of unioned s-categories can be found (we leave this as an exercise for the reader).

**Remark.** *These subcategories could instead be defined as sortings by themselves e.g. the opcartesian subcategory could be defined as in Definition 3.10 by requiring that  $\text{kind}(r)$  equals the union of controls of child-nodes and sorts of child-sites. However, considering these as subcategories eases some proofs of RPO creation and pushout reflection (particularly for full subcategories) and illustrates that they are merely restrictions of a more general sorting functor.*

**Remark.** *The prime product of two bigraphs in the subcategories of Definition 4.23 is defined only when there exists an interface sort which is a superset of the union of the outer interface sorts of the two bigraphs. The prime product of two bigraphs is always defined in the opcartesian and bounded complete subcategories. It is defined in a partitioned s-category when all roots of both bigraphs have the same sort.*

Although we concentrate on subcategories of kind sortings with visibility, the models of typed  $\lambda$ -calculi in Chapter 11 require us to consider two subcategories of kind sortings with semi-rigid capacities. We therefore define the following.

**Definition 4.24** (subcategories of sortings with semi-rigid capacities, interface kinds). *A full or fitting subcategory of a kind sorting with semi-rigid capacities is defined by defining the set  $\text{Int}$ . In the following, we define  $\text{Int}$  by defining a subset  $\text{Int}_{\mathcal{K}}$  of  $\mathcal{P}(\mathcal{K}_{\text{vis}})$  such that if  $S \in \text{Int}_{\mathcal{K}}$  then  $(n, f : \mathcal{K}_{\text{vis}} \rightarrow \mathbb{N}^{\odot}) \in \text{Int}$  if*

- $n \in \mathbb{N}^{\odot}$ ;
- $f(K) > 0$  if  $K \in S$ ,  $f(K) = 0$  if  $K \notin S$ ;
- and  $(n, f)$  is a maximum capacity interface sort.

<sup>7</sup>Milner previously proved that his homomorphic sortings reflect pushouts. Appendix A.2 compares homomorphic sorting with partitioned subcategories in order to provide a better explanation of the latter.



A full or fitting subcategory is:

- a meet s-category if  $Int_K$  is closed under intersection with itself and with  $Ctrl$ ;
- controlled if  $Ctrl \subseteq Int_K$ ;
- bounded complete if, given any non-empty set  $S$  of interface sorts which has an upper bound in  $(Int_K \cup Ctrl, \subseteq)$ , there exists a join for  $S$  in  $(Int_K, \subseteq)$ ;
- partitioned if distinct elements of  $Int_K$  are pairwise disjoint;
- fully partitioned if it is partitioned and each visible control is an element of some element of  $Int_K$ .

We call the set  $Int_K$  the interface kinds of the subcategory. A set of interface kinds defines a subcategory of a kind sorting with semi-rigid capacities.

The restriction on interface sorts both reduces the set of contexts and can constrain them to have an outer sort representing some sort or type (we demonstrate this for models of typed  $\lambda$ -calculi later). This in turn provides meaningful labels for the canonical labelled transition systems based on bounds or IPOs. Leaving invisible controls out of the interface sorts ensures that these controls are always tied to some other control *i.e.* they cannot exist on their own and can never be exposed to outer interfaces. We therefore say that they allow us to represent ‘hidden structure’ in the bigraphs of the system.

## 4.4 Conclusions

We use subcategories of kind sortings to model typed  $\lambda$ -calculi in bigraphs. We must consider whether these subcategories retain RPO creation and ask what level of pushout reflection they satisfy – the full s-category of kind sorted bigraphs creates RPOs but do not *strongly* reflect pushouts (besides the rigid sortings). Having identified various subcategories in Section 4.3, we prove/disprove these properties in the next chapter and consider a more suitable level of pushout reflection for kind sortings.

We observe the following of kind sortings. They are very free sortings *i.e.* they allow the sort of an interface to contain *any* subset of  $K_{vis}$  or capacity so long as the sorting rules are satisfied. The benefit of this freedom is that it allows a refined expression of absence of exposed nodes for parameters of reaction rules (see Chapter 10). The downside is that they are not immediately suitable for modelling sorted systems such as finite CCS or typed models of the  $\lambda$ -calculus. For

the latter applications, the subcategories introduced in this section allow the required amount of restriction; the sorting used by Milner to model finite CCS is a homomorphic subcategory and the models of the  $\lambda$ -calculus in Chapter 11 use partitioned and meet subcategories. In the next chapter we prove the safety of these subcategories with respect to the pure transition theory.

## 4.5 Related work

This chapter discusses the static theory of kind sortings. This presentation is based on Jensen and Milner's presentation of the static theory of pure bigraphs [74]. A notable omission here is an axiomatisation of kind sorted bigraphs. We believe that this is quite similar to Milner's axiomatisation of pure bigraphs [108] where we add verticals as elementary bigraphs.

Axiomatisations of sorted bigraphs have been given by Damgaard and Birkedal for binding bigraphs [45] and Grohmann and Miculan for directed bigraphs [65].

# Chapter 5

## Transition Theory

*Transdermal celebration*

*Caused a slight mutation*

*In the rift*

*It toppled down a nation*

*And left the people running for the hills*

*But the mutants that I see*

*Shine their beauty unto me*

*Transdermal Celebration – Ween*

In the previous and the following chapters, we concern ourselves with the static and reactive theories of kind sortings. Reactions are internal reconfigurations of agents of a system<sup>1</sup> and are not observable. However, the analysis and verification of process calculi and reactive systems is based on the *observable behaviour* of agents. These observations are made using behavioural equivalences on a labelled transition system, where the labels supply a context to an agent in which it reconfigures itself. In this chapter, we demonstrate how results from the transition theory of pure bigraphs transfer to kind sortings and certain subcategories thereof, in particular using previous results to prove the congruence of bisimilarity. We concentrate on the fundamental sorting here. The proofs for the sorting with visibility follow similarly and the proofs for the remaining sortings follow the same principles.

---

<sup>1</sup>Agents are ground bigraphs in the context of the dissertation.

We begin by recalling the definition of labels, reaction systems, and transition systems for bigraphs along with results by Jensen, Milner, and Leifer [74, 92] and Bundgaard and Sassone [24] which can be used to transfer useful properties of the transition theory along functors via proofs of RPO creation and pushout reflection. We then introduce a new notion of pushout reflection which lies between the current definitions as well as a corresponding transition system.

Sections 5.2–5.4 sketch the proofs of RPO creation and pushout reflection, using the notions of opcartesian arrows and jointly opcartesian cospans. Different properties are required of both constructions for the subcategories. For RPO creation, we need to be able to build an RPO interface (the interface  $\hat{I}$  at the center of an RPO  $\vec{B} : \rightarrow \hat{I}, B : \hat{I} \rightarrow$ ) whose sort is large enough to sort the legs  $\vec{B}$  of the RPO but small enough to fit inside nodes of the body  $B$ . For pushout reflection, the interface is given but we require its sort to be small enough to allow the mediating arrow between the pushout and any other bound.

Our conclusions for kind sortings are that bounded completeness of the poset  $(Int, \subseteq)$  of interface sorts of the subcategory suffices for RPO creation whereas a meet semi-lattice structure on  $(Int, \subseteq)$  suffices for pushout reflection if we only consider arrows with least sorts. The partitioned subcategories trivially satisfy both conditions and so have both properties, while the opcartesian subcategories have both properties as the interface sorts form a lattice  $(\mathcal{P}(\mathcal{K}_{vis}), \subseteq)$  and all arrows are opcartesian and hence have a least sort.

## 5.1 Transfer of dynamic theory

Process calculi are equipped with reaction rules (also called rewrite/reduction rules) which describe how subterms may interact or communicate with each other. Reaction is internal, describing how an agent/term  $a$  may reconfigure itself to become an agent  $a'$ . However, usually we are interested in how terms react with their surrounding environment *e.g.* how a mobile agent behaves in a certain context. Therefore, the notion of reaction is extended with labelled transitions  $a \xrightarrow{l} a'$  where the label  $l$  describes an interaction between  $a$  and its environment. Labels are thought of as observable actions (as opposed to reactions which are not observable) and can be used to reason about and to compare the behaviour of agents.

A labelled transition system (LTS) can be defined for a calculus with a set of transition rules over a set of labels. LTSs are considerably more useful for model checking and system analysis than the more easily defined reaction relation but they suffer from some drawbacks. Firstly, they are typically defined specific to one process calculus. Secondly, it is not always obvious what the set

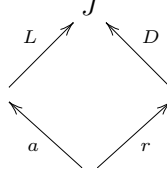


**Figure 5.1:** A labelled transition  $L$  of an agent  $a$  derived from a redex  $r$

of labels should be; care must be taken to ensure that the set of labels is both descriptive enough to capture all desired observations and small enough for tractable analysis.

The bigraph framework addresses these problems by introducing canonical definitions for labels. Labels in Brss are contexts of the systems *i.e.*  $a \xrightarrow{L} a'$  only if  $L \circ a$  reacts to form  $a'$ . However, we cannot consider all contexts as labels as analysis becomes intractable. This led to the question of which set of labels is small enough for analysis whilst retaining congruence of behavioural equivalences such as bisimilarity. An initial answer as to which labels to consider was given by Sewell for calculi without a naming structure [140]. Leifer and Milner then presented a definition of minimal labels for reactive systems with both nesting and linking [91, 90]. They observe that if we define terms and contexts as arrows of a category then the label above describes the commuting square of Figure 5.1 where  $r$  is a redex of a reaction rule of the system. Further, instead of considering all bounds  $L, D$  for an agent-redex pair  $a, r$ , we should only consider *minimal* bounds. This led to the consideration of the universal construction of relative pushout. The legs  $\vec{B}$  of a relative pushout for  $a, r$  relative to  $L, D$  is a minimal bound for  $a, r$ ; the relative pushout for  $a, r$  relative to  $\vec{B}$  is  $\vec{B}, \text{id}$ . This answers the question of which labels to consider; consider bounds  $L, D$  where the RPO for  $a, r$  relative to  $L, D$  is the triple  $L, D, \text{id}$ . These special RPOs are the idem pushouts defined in Section 2.2.

In his dissertation, Leifer showed that proofs of congruence for certain behavioural equivalences of reactive systems (a more general notion than bigraphs) are based primarily on the existence of sufficient RPOs [90]. Pure bigraphs have sufficient RPOs. Our task in this chapter is to show that our sortings retain at least enough of these constructions to allow proofs of bisimulation and safe reduction of the set of labels over certain transition systems and sets of reaction rules.

Before we discuss transitions, we summarise the definitions of reaction rules and bigraphical reactive systems. We omit details for presentation and refer the reader to Jensen and Milner's definitions [74, 114].

**Definition 5.1** (parametric reaction rule [114]). *A parametric reaction rule for bigraphs is a triple of the form  $(R : \langle m, X \rangle \rightarrow J, R' : \langle m', X' \rangle \rightarrow J, \eta)$  where  $R$  is the parametric redex,  $R'$  the parametric reactum, and  $\eta : m' \rightarrow m$  a map of ordinals.  $R$  and  $R'$  must be lean, and  $R$  must have no barren roots or idle names. The rule generates all ground reaction rules  $(r, r')$  where  $r \simeq (\text{id}_Y \otimes R) \circ d$ ,  $r' \simeq (\text{id}_Y \otimes R') \circ \bar{\eta}(d)$  and  $d : \langle m, Y \rangle$  is discrete. If  $\eta$  is injective, surjective, or bijective then the rule is respectively affine, total, or linear.*

A ground redex is generated from a parametric redex and a parameter  $d$ . Informally, the term  $\bar{\eta}(d)$  in the definition is generated from  $d$  by reordering, copying, and/or discarding regions of the parameter based on the map  $\eta$ ; the  $i^{\text{th}}$  region of  $\bar{\eta}(d)$  where  $i \in m'$  is equivalent to the  $\eta(i)^{\text{th}}$  region of  $d$ .

**Definition 5.2** (bigraphical reactive system [114]). *A (concrete) bigraphical reactive system (Brs) over a sorting  $\Sigma$  consists of an  $s$ -category of bigraphs over  $\Sigma$  equipped with a set  $\mathcal{R}$  of sorted parametric reaction rules closed under support equivalence ( $\simeq$ ). We denote it by  $\text{BIG}(\Sigma, \mathcal{R})$ .*

A Brs is safe if its sorting is safe (see Definition 5.15); it is affine, total, or linear when all the rules in  $\mathcal{R}$  are so.

**Definition 5.3** (simple, prime, basic [74, 114]). *A parametric redex is simple if every link is open, no site has a root as parent, and no two sites are siblings. A parametric redex is flat if no node has a node as parent. A reaction rule is simple, or prime, if its redex is so. A Brs is simple, or prime, if all its reaction rules are so and basic if all its reaction rules are flat, prime, simple, and no redex involves only a subset of the controls involved in another<sup>2</sup>.*

The following definition of transition is incomplete; we omit information regarding activity and location. This is dangerous [74] but is unimportant for the dissertation.

**Definition 5.4** (transition [74]). *A transition consists of a triple  $(a, L, a')$ , written  $a \xrightarrow{L} a'$ , where  $a$  and  $a'$  are ground bigraphs and there exists a ground reaction rule  $(r, r')$  and a context  $D$  such that the diagram of Figure 5.1 commutes and  $a' \simeq D \circ r'$ .*

The reaction rule and the diagram underlie the transition. A transition is minimal if the underlying diagram is an IPO and mono if  $L$  is a monomorphism.

The definition of transition system is more generally applicable. We consider it only for Brss here and specialise it to the full set of interfaces.

<sup>2</sup>We drop the requirement of freedom since we do not discuss binding bigraphs here.

**Definition 5.5** (transition system [74]). *Given a Brs  $\mathcal{B}IG(\Sigma, \mathcal{R})$ , a (labelled) transition system  $\mathcal{L}$  for  $\mathcal{B}IG(\Sigma, \mathcal{R})$  is given by a set  $\text{Trans}$  of transitions whose sources and targets are agents of  $\mathcal{L}$ .*

*The full (resp. standard) transition system  $\text{FT}$  (resp.  $\text{ST}$ ) for a Brs consists of all (resp. all minimal) transitions.*

**Definition 5.6** (wide bisimilarity [74]). *Let  $\mathcal{B}IG(\Sigma, \mathcal{R})$  be a Brs equipped with a transition system  $\mathcal{L}$ . A simulation (on  $\mathcal{L}$ ) is a binary relation  $S$  between agents with equal interface such that if  $a S b$  and  $a \xrightarrow{L} a'$  in  $\mathcal{L}$ , then whenever  $L \circ b$  is defined there exists  $b'$  such that  $b \xrightarrow{L} b'$  in  $\mathcal{L}$  and  $a' S b'$ . A bisimulation is a symmetric simulation. Then bisimilarity (on  $\mathcal{L}$ ), denoted by  $\sim_{\mathcal{L}}$ , is the largest bisimulation (on  $\mathcal{L}$ ).*

**Theorem 5.7** (congruence of bisimilarity [74]). *Wide bisimilarity of agents is a congruence in the standard transition system of any pure Brs.*

**Definition 5.8** (creating RPOs, reflecting pushouts [92]). *Let  $\mathcal{F}$  be any functor on an  $s$ -category  $\mathcal{A}$ . Then  $\mathcal{F}$  creates RPOs if, whenever  $\vec{D}$  bounds  $\vec{A}$  in  $\mathcal{A}$ , then any RPO for  $\mathcal{F}(\vec{A})$  relative to  $\mathcal{F}(\vec{D})$  has a unique  $\mathcal{F}$ -preimage that is an RPO for  $\vec{A}$  relative to  $\vec{D}$ .*

*$\mathcal{F}$  reflects pushouts if, whenever  $\vec{D}$  bounds  $\vec{A}$  in  $\mathcal{A}$  and  $\mathcal{F}(\vec{D})$  is a pushout for  $\mathcal{F}(\vec{A})$ , then  $\vec{D}$  is a pushout for  $\vec{A}$ .*

To prove *e.g.* bisimilarity over  $\text{ST}$ , it suffices to consider engaged transitions, transitions which participate in the underlying reaction.

**Definition 5.9** (engaged transitions [74]). *A transition of  $a$  in  $\text{ST}$  is said to be engaged if it can be based on a reaction with redex  $R$  such that  $a$  and  $R$  share nodes and edges.  $\text{PE}$  denotes the transition system of prime interfaces and engaged transitions.  $\mathcal{P}E$  denotes the sub-Lts in which the transitions are mono.*

Leifer and Milner [92, 110] proved the following theorem for link sortings<sup>3</sup> and place sortings respectively.  $\sim_{\text{ST}}$  denotes congruence for mono bisimilarity over  $\text{ST}$ ; this means that if  $a \sim_{\text{ST}} a'$  then  $C \circ a \sim_{\text{ST}} C \circ a'$  when  $C$  is mono. We will not define *adequacy* here; informally, it allows us to consider less transitions in proofs of bisimilarity. For some Brss, it suffices to consider the engaged transitions.

**Theorem 5.10** (useful sortings). *In a link/place-sorted Brs  $\mathcal{B}IG_h(\Sigma, \mathcal{R})$ :*

1. *If  $\Sigma$  creates RPOs then  $\sim_{\text{ST}}$  is a congruence and  $\sim_{\text{ST}}$  is a congruence for mono contexts.*

---

<sup>3</sup>The definition of link sorting is repeated in Section 6.2.

2. If in addition  $\Sigma$  reflects pushouts and  $\mathcal{R}$  is simple prime affine, then PE is adequate for ST and PE is adequate for ST.

This allows us to ensure properties of the dynamic theory of sortings based purely on properties of the sorting functor; the theory can be reflected along this functor. These results allow us to explore many generalisations of bigraphs whilst retaining a canonical labelled transition system where bisimilarity is a congruence. However, the condition that  $\Sigma$  reflects pushouts is stronger than necessary. Bundgaard and Sassone consider a weaker property.

**Definition 5.11** (weak pushout reflection [24]). *Let  $\mathcal{F}$  be any functor on an s-category  $\mathbf{A}$ .  $\mathcal{F}$  weakly reflects pushouts if, whenever  $\vec{D}$  is an IPO for  $\vec{A}$  in  $\mathbf{A}$  and  $\mathcal{F}(\vec{D})$  is a pushout for  $\mathcal{F}(\vec{A})$ , then  $\vec{D}$  is a pushout for  $\vec{A}$ .*

Bundgaard and Sassone showed that weak pushout reflection is sufficient to admit the adequacy theorem of Leifer and Milner. Weak pushout reflection is particularly useful for sortings with a notion of subsort *e.g.* in kind sortings.

**Theorem 5.12.** [24] *In a link/place-sorted Brs  $\mathbf{BIG}_h(\Sigma, \mathcal{R})$ , if  $\Sigma$  creates RPOs, weakly reflects pushouts, and  $\mathcal{R}$  is basic, then the prime engaged transitions are adequate for  $\sim_{ST}$ .*

This allows proofs of adequacy for a wider class of sortings. Our minor contribution here is to take this idea one step further and consider bounds with least sorts rather than IPOs. This property seems best captured with the notion of jointly opcartesian bounds that Birkedal et al. consider [17].

**Definition 5.13** (jointly opcartesian bounds reflect pushouts). *Given a functor  $\mathcal{F}$ , we say that jointly opcartesian bounds reflect pushouts along  $\mathcal{F}$  if, whenever a bound  $\vec{D}$  for  $\vec{A}$  in  $\mathbf{A}$  is jointly opcartesian and  $\mathcal{F}(\vec{D})$  is a pushout for  $\mathcal{F}(\vec{A})$ , then  $\vec{D}$  is a pushout for  $\vec{A}$ .*

In all kind sortings (and we believe this should hold for many if not all sortings), IPOs are jointly opcartesian bounds. Therefore, this definition generalises weak pushout reflection. Furthermore, while most kind sortings introduced here do not reflect pushouts, jointly opcartesian bounds along the sortings do reflect pushouts. It seems a useful property and lies between strong and weak pushout reflection. It also leads us to consider a new transition system which lies between FT and ST.

**Definition 5.14** (jointly opcartesian bound transition system (JOT)). *A transition is jointly opcartesian if the cospan in its underlying diagram is jointly opcartesian. The jointly opcartesian transition system JOT for a sorting consists of all jointly opcartesian transitions.*



JOT seems the natural generalisation of FT to sortings where jointly opcartesian bounds represent least sorts and reflect pushouts.

We weaken the following definition to only require that jointly opcartesian bounds reflect pushouts rather than arbitrary bounds as we concentrate on the weaker property throughout.

**Definition 5.15** (safe functor, safe sorting [114]). *A functor  $\mathcal{F} : \mathbf{A} \rightarrow \mathbf{B}$  of s-categories is safe if it creates RPOs and jointly opcartesian bounds reflect pushouts.*

*A sorting discipline is safe if its sorting functor is safe.*

Finally, we introduce a simple observation that to show strong pushout reflection, we do not need to consider all bounds, but only those which are the preimages of pushouts.

**Definition 5.16** (sufficiently jointly opcartesian). *Let  $\mathcal{U} : \mathbf{D} \rightarrow \mathbf{C}$  be a sorting functor and all pushouts in  $\mathbf{C}$  satisfy some condition  $\Phi$ . If  $\Phi$  is reflected by  $\mathcal{U}$  and all cospans of  $\mathbf{D}$  which satisfy  $\Phi$  are jointly opcartesian with respect to  $\mathcal{U}$  then we say that  $\mathbf{D}$  is sufficiently jointly opcartesian with respect to  $\mathcal{U}$ .*

**Proposition 5.17.** *If  $\mathbf{D}$  is sufficiently jointly opcartesian with respect to  $\mathcal{U} : \mathbf{D} \rightarrow \mathbf{C}$  then  $\mathcal{U}$  reflects pushouts.*

*Proof.* Let  $\vec{B}$  be a bound for  $\vec{A}$  in  $\mathbf{D}$  such that  $\vec{B}^u$  is a pushout for  $\vec{A}^u$  in  $\mathbf{C}$ . Let  $\vec{C}$  be another bound for  $\vec{A}$  in  $\mathbf{D}$ . As  $\vec{B}^u$  is a pushout, there exists a unique mediator  $j'$  from  $\vec{B}^u$  to  $\vec{C}^u$  in  $\mathbf{C}$ . Furthermore,  $\vec{B}^u$  satisfies some condition  $\Phi$ . Therefore,  $\vec{B}$  satisfies  $\Phi$  and hence  $\vec{B}$  is jointly opcartesian with respect to  $\mathcal{U}$ . Therefore, there exists a mediator  $j$  from  $\vec{B}$  to  $\vec{C}$  in  $\mathbf{D}$  which is unique as  $\mathcal{U}$  is faithful.  $\square$

As an example of such a property  $\Phi$ , if  $\vec{B} : \vec{J} \rightarrow K$  is a pushout for  $\vec{A} : I \rightarrow \vec{J}$  in an s-category of pure or local bigraphs then no root (resp. name) of  $K$  is barren (resp. idle) in both  $B_0$  and  $B_1$ . Therefore, in order to prove pushout reflection for a sorting functor into an s-category of pure or local bigraphs which reflects barrenness and idleness<sup>4</sup>, it suffices to prove that all cospans where no root is barren and no name is idle (in both bigraphs) are jointly opcartesian.

Having hopefully impressed the relevance of RPO creation and pushout reflection on the reader, we proceed with our investigation.

---

<sup>4</sup>This seems typical.

## 5.2 Basic properties

In this section, we present basic properties of kind sortings, the subcategories of kind sortings with visibility given in Definition 4.23, and the subcategories of sortings with semi-rigid capacities given in Definition 4.24. These properties are used in the proofs of RPO creation and pushout reflection. When we present properties of subcategories without qualifying the sorting, the reader can assume we are discussing kind sortings with visibility. Most of the proofs are contained in Appendix A.3 or generalise previous work [118].

**Lemma 5.18.** *A cospan  $\vec{A} : \vec{H} \rightarrow I$  of a kind sorted s-category is jointly opcartesian with respect to the sorting functor iff the sort of each place of  $I$  has the least outer sort which satisfies the sorting conditions for  $\vec{A}$ .*

*Proof.* See Appendix A.3, Lemma A.13. □

**Corollary 5.19.** *All cospans of kind sorted s-categories are nearly jointly opcartesian.*

*Proof.* Each cospan  $\vec{A} : \vec{H} \rightarrow J$  of a kind sorted s-category can be decomposed as  $A_i = J \uparrow I \circ A'_i$  where each place of  $I$  has the least sort which sorts both  $A'_0$  and  $A'_1$ . □

**Corollary 5.20.** *An arrow  $A : H \rightarrow I$  of a kind sorted s-category is opcartesian with respect to the sorting functor iff the sort of each place of  $I$  has the least outer sort which satisfies the sorting conditions for  $A$ .*

*Proof.*

$\Rightarrow$  Similar to the proof of Lemma A.13 on page 7 of the appendix.

$\Leftarrow$  Let  $D : H \rightarrow J$  be some arrow such that  $D^u = B' \circ A^u$  for some  $B'$ . Consider the cospans  $(A, A)$  and  $(D, D)$ . The sort of  $I$  has the least outer sort which satisfies the sorting conditions for  $A$  and hence  $(A, A)$ . By Lemma 5.18, there exists a unique mediator  $B$  from  $(A, A)$  to  $(D, D)$ . Therefore,  $B$  is the unique arrow such that  $D = B \circ A$ . □

**Corollary 5.21.** *All arrows of kind sorted s-categories are nearly opcartesian.*

*Proof.* Each arrow  $A : H \rightarrow J$  of a kind sorted s-category can be decomposed as  $A = J \uparrow I \circ A'$  where each place in the outer interface of  $A' : H \rightarrow I$  has a least sort. □

**Proposition 5.22** (opcartesian redexes imply opcartesian labels in JOT of a kind sorted Brs). *Let  $(B_0, B_1)$  be a jointly opcartesian bound for  $(A_0, A_1)$  and  $A_1$  be opcartesian. Then  $B_0$  is opcartesian.*

*Proof.* See Appendix A.3, Proposition A.14.  $\square$

This proof is useful when  $A_1$  is a ground redex. Then if the redexes of reaction rules of a kind sorted Brs are opcartesian and we only consider opcartesian parameters to the rules, all labels in the JOT transition system will also be opcartesian.

**Lemma 5.23.** *The pure functor of a subcategory  $\mathcal{I}$  of a kind sorting  $\mathcal{U}$  is faithful.*

**Lemma 5.24.** *All arrows of fitting meet subcategories of kind sortings are opcartesian with respect to their pure functor.*

*Proof.* See Appendix A.3, Lemma A.15.  $\square$

**Lemma 5.25.** *Let  $D = B \circ A$  and  $D$  and  $A$  be opcartesian arrows of a kind sorted  $s$ -category with respect to the sorting functor. Then  $B$  is opcartesian.*

*Proof.* See Appendix A.3, Lemma A.16.  $\square$

**Corollary 5.26.** *All arrows of opcartesian subcategories are opcartesian.*

*Proof.* Let  $A : H \rightarrow I$  be an arrow of the opcartesian subcategory of  $\mathcal{BIG}(\Sigma_K)$  where  $D : H \rightarrow J$  and where  $B' : I^u \rightarrow J^u$  is a pure bigraph such that  $D^u = B' \circ A^u$ . By Corollary 5.20, there exists such a unique bigraph  $B$  in  $\mathcal{BIG}(\Sigma_K)$  such that  $\mathcal{U}(B) = B'$ . By Lemma 5.25,  $B$  is an arrow of the opcartesian subcategory.  $\square$

Hence, opcartesian and meet subcategories are weak opfibrations.

**Lemma 5.27.** *Let  $G$  be a bigraph of a partitioned subcategory. For any root  $r$  of  $G$ , if  $G(s) = r$  for some site of  $G$  then  $\text{kind}(s) = \text{kind}(r)$ .*

*Proof.* By condition KV2,  $\text{kind}(s) \subseteq \text{kind}(r)$ . As the interface sorts are disjoint,  $\text{kind}(s) = \text{kind}(r)$ .  $\square$

## 5.3 Relative pushout creation

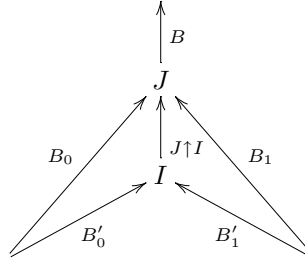
In this section, we sketch proofs of RPO creation for the kind sortings of Chapter 3. A proof for the fundamental kind sorting was previously published [118] and the remaining proofs follow the same structure. We then give proofs of RPO creation (when they exist) for the subcategories of kind sortings introduced in Chapter 4.

### 5.3.1 Relative pushouts for kind sortings

The RPO construction for the fundamental sorting and sorting with visibility is given in Construction A.17 on page 10 of the appendices.

**Lemma 5.28.** *Let  $(\vec{B} : \rightarrow \vec{J}, B)$  be a relative pushout for  $\vec{A}$  relative to  $\vec{D}$ . Then  $\vec{B}$  is jointly opcartesian with respect to the sorting functor.*

*Proof.* Assume that  $\vec{B}$  is not jointly opcartesian. By Lemma 5.18, there exists a pair  $\vec{B}' : \rightarrow \vec{I}$  and a vertical  $J \uparrow I$  as in the diagram below. Then  $\vec{B}'$  is a bound for  $\vec{A}$  relative to  $\vec{D}$ . However, by the same reasoning as Lemma 5.18, there is no mediator from  $(\vec{B}, B)$  to  $(\vec{B}', B \circ J \uparrow I)$ , contradicting the assumption that  $(\vec{B}, B)$  is an RPO.



□

**Corollary 5.29.** *All IPOs are jointly opcartesian with respect to the sorting functor.*

It follows that the RPO constructions for kind sortings hinge on the definition of the least sort for a cospan given in Definition 4.6. RPO creation is proven as follows, following the same steps as Jensen and Milner's proof for binding bigraphs [74].

First, we define an RPO construction. In all our sortings, we take a pure RPO, define the sorting of the interface which by Lemma 5.28 is given by Definition 4.6, and then prove the construction is sorted. This is done by proving that the legs of the RPO are sorted and then applying Lemma 5.18 to prove that  $B$  is sorted.

Next, we prove that the construction is indeed an RPO. This follows from a simple argument using Lemma 5.18, the uniqueness of the underlying RPO, and the faithfulness of the sorting functor which proves that a unique mediator exists between the construction and any other relative bound.

The proof for fundamental kind sortings has previously been given [118] and is reproduced in Appendix A.4. The proofs for the other kind sortings follow similarly.

**Corollary 5.30.** *Kind sortings preserve RPOs.*

*Proof.* The proof is identical to [74, Corollary 11.6].  $\square$

The consistency conditions for spans of kind sortings *i.e.* the conditions which are necessary and sufficient for span to have a bound, follow the conditions for pure bigraphs besides a small addition regarding the sorting of roots of the span. The following was previously proven for the fundamental kind sorting [118] but generalises easily.

**Proposition 5.31.** *A span  $A_0 : I \rightarrow \langle m_0, \vec{\theta}^0, X \rangle$ ,  $A_1 : I \rightarrow \langle m_1, \vec{\theta}^1, X \rangle$  of bigraphs in a kind sorted  $s$ -category is consistent iff its underlying pure span is consistent and if it satisfies the condition:*

**KC** *if  $A_i(w_2) \in V_i - V_2$  then  $A_i(w_2) = r \in m_i$  and  $\theta_r^i \preceq \text{kind}(A_i(w_2))$ .*

where  $V_i$  is the set of nodes of  $A_i$ ,  $V_2$  is the set of shared places (sites or nodes) of the span,  $w_2 \in V_2$ , and  $\preceq$  is the partial order of Definition 4.7.

An argument based on the proposition shows that bounds are not reflected by the sorting functor in general. The following theorems are similar to those for binding bigraphs [74].

**Theorem 5.32.** *Let  $\vec{A}$  satisfy the consistency conditions and  $\vec{A}^u$  have a pure IPO  $\vec{B}'$ . Then  $\vec{A}$  has a kind IPO  $\vec{B}$ , with  $\vec{B}^u = \vec{B}'$ .*

*Proof.* The consistency conditions for kind sorted bigraphs imply the consistency conditions for pure bigraphs so  $\vec{A}^u$  has at least one IPO  $\vec{B}'$ . We construct a bound  $\vec{B}$  such that  $\vec{B}^u = \vec{B}'$  by defining the outer interface to have the least sorting. Then  $(\vec{B}^u, \text{id})$  is an RPO for  $\vec{A}^u$  to  $\vec{B}^u$ . Using the argument of Proposition A.18,  $(\vec{B}, \text{id})$  is an RPO for  $\vec{A}$  to  $\vec{B}$  and hence a kind IPO for  $\vec{A}$ .  $\square$

**Theorem 5.33.** *If  $\vec{A}$  has a kind IPO  $\vec{B}$ , then  $\vec{A}^u$  has a pure IPO  $\vec{B}^u$ .*

*Proof.* This is a special case of Corollary 5.30.  $\square$

By Theorems 5.32 and 5.33, when a pair  $\vec{A}$  of kind place graphs is consistent, there is a precise correspondence between its kind IPOs and the pure IPOs of  $\vec{A}^u$ . By Corollary 5.29, a kind sorted IPO can be constructed for a consistent span  $\vec{A}$  by constructing the underlying pure IPO and assigning the least sorting to its outer interface.

### 5.3.2 Relative pushouts for subcategories of kind sortings

Fitting subcategories do not create RPOs in general and neither do unioned fitting subcategories – we have given counter-examples for both in previous work [123] – so we need to examine which subcategories retain enough of these universal constructions. However, unioned *full* subcategories

do create RPOs (Proposition 5.36) and as we believe they weakly reflect pushouts, they seem preferable to their unioned fitting counterparts.

To simplify the proof of RPO creation, we use the following proposition.

**Proposition 5.34.** *If  $\mathcal{F} : \mathbf{A} \rightarrow \mathbf{B}$  and  $\mathcal{G} : \mathbf{B} \rightarrow \mathbf{C}$  create RPOs then  $\mathcal{GF}$  creates RPOs.*

*Proof.* See Appendix A.4, Proposition A.19. □

Therefore, to prove that the pure functor of a subcategory  $\mathcal{I}$  of a sorting  $\mathcal{U}$  creates RPOs, it suffices to prove that  $\mathcal{I}$  creates RPOs.

**Lemma 5.35.** *Let  $\mathcal{I} : \mathbf{C} \rightarrow \mathbf{D}$  be a full subcategory. If  $\vec{B}', B'$  is a relative bound for  $\mathcal{I}(\vec{A})$  relative to  $\mathcal{I}(\vec{D})$  and the common interface  $J$  of  $\vec{B}', B'$  is an object of  $\mathbf{C}$  then  $\vec{B}', B'$  has a unique preimage  $\vec{B}, B$  along  $\mathcal{I}$ . Furthermore, if  $\vec{B}', B'$  is an RPO for  $\mathcal{I}(\vec{A})$  relative to  $\mathcal{I}(\vec{D})$  then  $\vec{B}, B$  is an RPO for  $\vec{A}$  relative to  $\vec{D}$ .*

*Proof.* Both properties follows from the fullness and faithfulness of  $\mathcal{I}$ . □

The constructions for relative pushouts in downward closed and controlled fitting subcategories, downward closed and unioned fitting subcategories, unioned full subcategories, and partitioned s-categories is given in Appendix A.4 as are the proofs of the following propositions.

**Proposition 5.36** (creation of RPOs). *Whenever  $\vec{D}$  bounds  $\vec{A}$  in a subcategory  $\mathcal{I}$  of a kind sorting then any RPO for  $\mathcal{U}(\vec{A})$  relative to  $\mathcal{U}(\vec{D})$  has a unique  $\mathcal{U}$ -preimage that is an RPO for  $\vec{A}$  relative to  $\vec{D}$  if:*

1.  $\mathcal{I}$  is a downward closed and controlled fitting subcategory;
2.  $\mathcal{I}$  is a downward closed and unioned fitting subcategory;
3.  $\mathcal{I}$  is a unioned full subcategory;
4.  $\mathcal{I}$  is a partitioned subcategory.

**Proposition 5.37.** *Bounded complete and controlled full subcategories of kind sortings create RPOs along their pure functor.*

## 5.4 Pushout reflection

Pushout reflection does not hold in general for most of the kind sortings (the rigid sortings are exceptional). Counter-examples are easily found for bounds which are not jointly opcartesian. However, we can prove that jointly opcartesian bounds reflect pushouts, a property which lies between (strong) pushout reflection and weak pushout reflection. This seems the natural result for sortings with subsorts and is strong enough for Theorem 5.12 to apply. We then prove that either bounds or jointly opcartesian bounds reflect pushouts for some of the subcategories of kind sortings.

### 5.4.1 Pushout reflection for kind sortings

**Proposition 5.38** (jointly opcartesian bounds reflect pushouts along kind sortings). *If  $\vec{B}$  is a jointly opcartesian bound for  $\vec{A}$  along a kind sorting  $\mathcal{U}$  and  $\mathcal{U}(\vec{B})$  is a pushout for  $\mathcal{U}(\vec{A})$ , then  $\vec{B}$  is a pushout for  $\vec{A}$ .*

*Proof.* See Appendix A.5, Proposition A.23. □

**Corollary 5.39.** *Kind sortings weakly reflect pushouts.*

*Proof.* By Proposition 5.38 and Corollary 5.29. □

### 5.4.2 Pushout reflection for subcategories of kind sortings

As with RPO creation, fitting subcategories do not reflect pushouts in general. We previously reasoned that the problem lies when roots are identified in the pushout which requires the sort to be ‘increased’; we cannot guarantee the required properties of this new sort in general [123]. Arbitrary choices of subcategory lead to another counterexample of pushout reflection. For example, let  $\theta$  be a least outer sort for a cospan  $\vec{G}$  of prime kind sorted bigraphs where  $\vec{G}^u$  is a pushout for some span. If we consider a subcategory of the sorting which only contains the interface sorts  $\theta \cup \{K\}$  and  $\theta \cup \{L\}$ ,  $L \neq K$  then pushout reflection is broken as neither sort subsumes the other.

These counterexamples led to the definition of meet subcategories where the partial order  $(Int, \subseteq)$  on interface sorts is a meet-semilattice and which have strong pushout reflection<sup>5</sup>.

**Lemma 5.40.** *Any full subcategory  $\mathcal{I} : \mathcal{C} \rightarrow \mathcal{D}$  reflects pushouts.*

*Proof.* Follows from the fullness and faithfulness of  $\mathcal{I}$ . □

---

<sup>5</sup>We conjecture that having bounded meets is sufficient for strong pushout reflection.

In general, pure functors  $\mathcal{UI}$  of subcategories  $\mathcal{I}$  of kind sortings  $\mathcal{U}$  may not reflect pushouts *e.g.* the pure functor  $\mathcal{UI}$  of an unioned full subcategory does not reflect pushouts, but we state the following conjecture.

**Conjecture 5.41.** *If  $\vec{B}$  is a jointly opcartesian bound for  $\vec{A}$  along  $\mathcal{UI}$  where  $\mathcal{I}$  is a unioned full subcategory and  $\mathcal{UI}(\vec{B})$  is a pushout for  $\mathcal{UI}(\vec{A})$ , then  $\vec{B}$  is a pushout for  $\vec{A}$ .*

Both meet fitting subcategories and partitioned subcategories reflect pushouts whereas meet full subcategories only reflect pushouts on jointly opcartesian bounds.

**Proposition 5.42.** *Whenever  $\vec{D}$  bounds  $\vec{A}$  in a subcategory  $\mathcal{I}$  of a kind sorting and  $\mathcal{U}(\vec{D})$  is a pushout for  $\mathcal{U}(\vec{A})$ , then  $\vec{D}$  is a pushout for  $\vec{A}$  if:*

1.  $\mathcal{I}$  is a meet fitting subcategory;
2.  $\mathcal{I}$  is a partitioned subcategory.

*Proof.* See Appendix A.5, Proposition A.24. □

**Lemma 5.43.** *All cospans in a meet full subcategory are nearly jointly opcartesian.*

**Corollary 5.44.** *IPOs in a controlled meet full subcategory are jointly opcartesian.*

*Proof.* Proof by contradiction using sorting condition K2 and the previous lemma. □

**Proposition 5.45.** *If  $\vec{B}$  is a jointly opcartesian bound for  $\vec{A}$  along  $\mathcal{UI}$  where  $\mathcal{I}$  is a controlled meet full subcategory and  $\mathcal{UI}(\vec{B})$  is a pushout for  $\mathcal{UI}(\vec{A})$ , then  $\vec{B}$  is a pushout for  $\vec{A}$ .*

*Proof.* See Appendix A.5, Corollary A.25. □

**Corollary 5.46** (controlled meet full subcategories weakly reflect pushouts). *If  $\vec{B}$  is an IPO for  $\vec{A}$  in a controlled meet full subcategory  $\mathcal{I} : \mathbf{A} \rightarrow \mathbf{BIG}(\Sigma_K)$  and  $\mathcal{UI}(\vec{B})$  is a pushout for  $\mathcal{UI}(\vec{A})$ , then  $\vec{B}$  is a pushout for  $\vec{A}$ .*

*Proof.* Follows from Corollary 5.44 and Proposition 5.45. □

## 5.5 Conclusions

This chapter had three main contributions.

Firstly, we defined the JOT transition system and considered pushout reflection for jointly opcartesian bounds. This latter notion lies between strong and weak pushout reflection and holds for



| Kind sorting $\mathcal{U}$ | Creates<br>RPOs | Reflects<br>pushouts | JOBs reflect<br>pushouts | Axiomatisation<br>is neat |
|----------------------------|-----------------|----------------------|--------------------------|---------------------------|
| Fundamental                | ✓               | ✗                    | ✓                        | ✓                         |
| With visibility            | ✓               | ✗                    | ✓                        | ✓                         |
| Rigid max capacities       | ✓               | ✓                    | ✓                        | ✓                         |
| Semi-rigid max capacities  | ✓               | ✗                    | ✓                        | ✓                         |
| Max capacities             | ✓               | ✗                    | ✓                        | ✓                         |
| Rigid min-max capacities   | ✓               | ✓                    | ✓                        | ✗                         |

**Table 5.1:** *Properties of kind sortings*

most of our sortings and subcategories. We also introduced the idea that a functor need only be sufficiently jointly opcartesian; that is, not all JOBs need reflect pushouts but only those whose image under the sorting functor may be a pushout. This is particularly useful in sortings whose cospans are not jointly opcartesian; this can occur quite easily with link sortings where decomposition of bigraphs can break the sorting due to the introduction of idle names. As decomposition does not respect the sorting, pushout reflection does not immediately follow. In the next chapter, we discuss plain link sorting where decomposition breaks the sorting and use Proposition 5.17 to prove that pushouts are still reflected.

Secondly, we demonstrated the usefulness of our kind sortings *i.e.* that they retain the congruence and adequacy properties of Theorems 5.10 and 5.12, by giving proofs of RPO creation and pushout reflection.

Thirdly, we investigated subcategories of kind sortings with visibilities. This led us to realise that proofs of RPO creation and pushout reflection for our sortings depend on the structure of the partially ordered set  $(Int, \subseteq)$  of interface sorts. When this poset has a meet-semilattice structure then pushout reflection holds for jointly opcartesian bounds in full subcategories and for arbitrary bounds in fitting subcategories whereas when the poset is bounded complete, RPO creation holds for full subcategories. Our findings are collected in Tables 5.1 and 5.2 where the claim of neat axiomatisations is currently conjecture.

Before we discuss other sortings in the next chapter, we remark that although strong pushout reflection does not hold in general for kind sortings, this only becomes of concern when we consider

|         | Subcategory $\mathcal{I}$            | $\mathcal{UI}$ creates RPOs | $\mathcal{UI}$ reflects POs |
|---------|--------------------------------------|-----------------------------|-----------------------------|
| full    | unioned                              | ✓                           | maybe for JOBs              |
|         | bounded complete and controlled      | ✓                           | maybe for JOBs              |
|         | bounded complete and controlled meet | ✓                           | ✓ for JOBs                  |
| fitting | meet                                 | ✗                           | ✓                           |
|         | downward closed                      | ✗                           | ✓                           |
|         | – and controlled                     | ✓                           | ✓                           |
|         | – and unioned                        | ✓                           | ✓                           |
|         | opcartesian                          | ✓ [118]                     | ✓ [118]                     |
|         | partitioned                          | ✓                           | ✓                           |

**Table 5.2:** Proven properties of subcategories of kind sortings with visibility

labelled transition systems which are not based on idem pushouts. In that case, we believe that the property of jointly opcartesian bounds reflecting pushouts should suffice; this amounts to requiring that if  $L$  is a labelled transition of an agent  $a$  with respect to a ground reaction rule  $r$  and a context  $D$  for the rule, then the shared outer interface  $J$  of  $L$  and  $D$  has a least sorting for the pair (see Figure 5.1). This seems reasonable; it is not clear why we would not wish to use the least sorting.

We have the following corollaries of Theorems 5.10 and 5.12.

**Corollary 5.47.** *Bisimilarity (resp. mono bisimilarity) over the standard transition system in a kind sorted Brs of hard bigraphs  $\mathcal{BIG}_h(\Sigma_K, \mathcal{R})$ , is a congruence (resp. is a congruence for mono contexts).*

*If in addition  $\mathcal{R}$  is simple prime affine, then the prime engaged transitions are adequate for  $\sim_{ST}$ .*

**Corollary 5.48.** *Bisimilarity (resp. mono bisimilarity) over the standard transition system in a subcategory of a kind sorted Brs of hard bigraphs  $\mathcal{BIG}_h(\Sigma_K, \mathcal{R})$ , is a congruence (resp. is a congruence for mono contexts) if the subcategory creates RPOs according to Table 5.2.*

*If in addition  $\mathcal{R}$  is simple prime affine and jointly opcartesian bounds in the subcategory reflect pushouts according to Table 5.2 then the prime engaged transitions are adequate for  $\sim_{ST}$ .*

### 5.5.1 Related work

Labels of transition systems for bigraphs are based on relative pushouts. RPOs are a general, categorical construction and were initially studied in the more general setting of wide reactive systems [91]. Sassone and Sobociński have studied the related notion of groupoid RPOs to derive labels in a 2-categorical setting [137].

Jensen’s safety conditions for sortings [73] are sufficient to prove RPO creation. However, they cannot be immediately applied to kind sortings due to the third condition which concerns the decomposability of a bigraph along a sorting. It requires that a kind sorted bigraph can be decomposed at any nesting level. This is not possible in the presence of invisible nodes as these cannot be exposed to the outer interface. However, we believe that Jensen’s conditions and proof can – with only a slight modification – be adapted to prove RPO creation for kind sortings.

Birkedal, Debois, and Hildebrandt [16] have studied Jensen’s results within the setting of obfibrations. They prove that if a sorting of pure/local bigraphs is a weak opfibration, reflects prefixes, and has vertical pushouts, then the sorting creates RPOs<sup>6</sup>. Kind sortings are weak obfibrations and reflect prefixes when the signature does not contain invisible controls. They also have vertical pushouts; this follows easily as the fibres form a lattice (for kind sortings with visibility this lattice is isomorphic to  $\mathcal{P}(\mathcal{K}_{\text{vis}})$ ) and pushouts in the lattice are pushouts in the sorted s-category by the RPO constructions. Therefore, Birkedal, Debois, and Hildebrandt’s results also provide a proof of RPO creation in the absence of invisible controls in the signature<sup>7</sup>. Their approach has inspired many of our definitions, approaches, and results in this chapter.

---

<sup>6</sup>The verticals above an identity  $\text{id}_A$  form a category called the fibre over  $A$ . A functor  $\mathcal{U} : \mathcal{E} \rightarrow \mathcal{B}$  is said to have vertical pushouts if and only if the fibres have pushouts and such pushouts are also pushouts in  $\mathcal{E}$  [17].

<sup>7</sup>Prefix reflection is similar to Jensen’s decomposability condition and the existence of invisible controls will break this property. Similarly, not all sub-s-categories of kind sortings (regardless of the existence of invisible controls) reflect prefixes along  $\mathcal{UI}$  as an interface which sorts the decomposition may not exist in the subcategory.

# Chapter 6

## Further Sortings

*Class A*

*Class B*

*Is that the only chemistry between us?*

*The Chemistry Between Us – Suede*

This chapter discusses link sortings, introduces ideas for future sortings, and presents sortings which will be used to demonstrate applications of kind sortings in the final part of the dissertation. In particular, we present a sorting to remove junk bigraphs from  $\Lambda\text{BIG}$ .

We begin in Section 6.1 by considering kind sorting where not only the parent-child relationship is sorted, but where the entire ‘family tree’ is sorted. This may seem a special case of kind sorting but while the containment relation is limited, the sorts of places in these bigraphs are more expressive. In a similar vein, we consider capacities which limit the number of nodes which can be located anywhere below a node in a ground bigraph. We do not investigate the properties of these sortings here but they seem close enough to kind sortings that we conjecture RPO creation and opcartesian pushout reflection to hold.

Leifer and Milner’s definition of link sorting is repeated in Section 6.2. We then introduce our *tiled link sorting* which was derived to model tile-based games where the tiles have an even number of sides as bigraphical systems. These sorted bigraphs have undirected points like pure bigraphs which may link together unconditionally. They also have directed points, which may only link together in pairs of opposite directions. This seems to combine a generalisation of Leifer and

Milner’s *directed linear* link graphs [92] with pure link graphs. Next, we describe the simple *plain sorting* we use in the models of typed  $\lambda$ -calculi in Section 11.1. We prove both these link sortings safe.

In Section 6.3 we discuss composition of sortings and a simple pairing of sortings which combines two sortings without mixing their properties. The latter is a special case of a notion presented by Birkedal et al. [16]. We prove that this form of combination (essentially a pullback) is safe and use it to combine kind sorting with visibility and rigid control-sorting [15]. This combination is used to present a ‘bigraphical algorithm’ in Chapter 10 which is based on related work [14].

## 6.1 Place sortings

In this section we present ideas for sortings similar to kind sortings but which consider the relationship between a node and all nodes below it rather than just the parent-child relationship.

### 6.1.1 Deep kind sorting

This is a variation on the fundamental kind sorting. Rather than stating which controls a  $K$ -node may parent, we state which controls a  $K$ -node may be an ancestor of (hence ‘deep’). A fundamental kind sorting describes a relation on controls;  $K \sim L$  iff  $L \in \text{kind}(K)$  *i.e.*  $K$  is related to controls it may contain. However, to ensure that composition preserves the new sorting, we will require that deep kind sorting describe a *transitive* relation *i.e.* if also  $L$  can contain  $M$ -nodes (so  $K \sim L$  and  $L \sim M$ ) then  $K$  must be able to contain them as well ( $K \sim M$ ).

**Definition 6.1** (deep kind signature). *A deep kind signature  $\{\mathcal{K}, \text{arity}, \text{actv}, \text{kind}\}$  is composed of a set  $\mathcal{K}$  of controls and three maps:*

$$\begin{aligned} \text{arity} : \mathcal{K} &\rightarrow \mathbb{N} & \text{kind} : \mathcal{K} &\rightarrow \mathcal{P}(\mathcal{K}) \\ \text{actv} : \mathcal{K} &\rightarrow \{\text{passive}, \text{active}\} \end{aligned}$$

*such that if  $K \in \text{kind}(L)$  then  $\text{kind}(K) \subseteq \text{kind}(L)$ . We call  $\text{kind}(K)$  the kind or sort of  $K$ .*

Because of the condition in the definition, deep kind sorting is less general/expressive than kind sorting for modelling containment relationships. However as we will see, the deep sorting is interesting for other reasons.

As an aside, if we consider the directed graph with the set of nodes  $\mathcal{K}$  and arrows characterised by  $\{(L \rightarrow K) \mid L \in \text{kind}(K)\}$ , then cycles in the graph contain nodes of equal sort due to the transitivity mentioned above.

**Definition 6.2** (deep kind sorting). A place-sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  over a deep kind signature  $\mathcal{K}$  is a deep kind sorting if  $\Theta = \mathcal{P}(\mathcal{K})$  and  $\Phi$  requires for all bigraphs  $G$  that:

**K1** if  $p = G(v)$  then  $\text{ctrl}(v) \cup \text{kind}(v) \subseteq \text{kind}(p)$ ;

**K2** if  $p = G(s)$  then  $\text{kind}(s) \subseteq \text{kind}(p)$ ;

**K3** if  $\text{kind}(v) = \emptyset$ ,  $v$  parents no sites;

where  $p$  is a root or node,  $s$  a site,  $v$  is a node, and the sort of a place  $r$  is written  $\text{kind}(r)$ .

We can describe this more succinctly by letting  $\text{ctrl}(s) = \emptyset$  for all sites. The definition below explains the sorting more clearly but we conjecture that it is more expensive to automatically verify.

**Definition 6.3** (deep kind sorting). A place-sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  over a deep kind signature  $\mathcal{K}$  is a deep kind sorting if  $\Theta = \mathcal{P}(\mathcal{K})$  and  $\Phi$  requires for all bigraphs  $G$  that:

**K1** if  $w <_G p$  then  $\text{ctrl}(w) \cup \text{kind}(w) \subseteq \text{kind}(p)$ ;

**K2** if  $\text{kind}(v) = \emptyset$ ,  $v$  parents no sites;

where  $p$  is a root or node,  $w$  is a site or node,  $v$  is a node.

It may help to visualise deep kind sorting by noting that every deep kind sorted bigraph gives rise to a monotonic function between the parent function of the bigraph and the signature set. Define  $\text{ctrl}(s) = \emptyset$  for all interface places, let  $P_G$  denote the nodes, sites, and roots of a bigraph  $G$  and let  $(P_G, <_G)$  denote the order on nodes and places of a bigraph  $G$  defined by the *prnt* map. We have a monotonic function  $\text{ctrl}(-) \cup \text{kind}(-) : (P_G, <_G) \rightarrow (\mathcal{P}(\mathcal{K}), \subseteq)$  for every deep kind sorted bigraph  $G$ .

**Example 6.4.** Let the signature  $\mathcal{K} = \{K, L, M, N\}$ . Let *kind* be defined as:

$$K \mapsto \{K, L, M\}, \quad L \mapsto \emptyset, \quad M \mapsto \{L, N\}, \quad N \mapsto \emptyset.$$

Let  $G : \langle 2, (\{K, L\}\{L\}) \rangle \rightarrow \langle 1, (\{K, L, M, N\}) \rangle$  be the deep kind sorted bigraph whose place graph is depicted in Figure 6.1 where  $\text{ctrl}_G$  is defined as:

$$v_0 \mapsto K, \quad v_1 \mapsto M, \quad v_2 \mapsto L, \quad v_3 \mapsto L, \quad v_4 \mapsto L.$$

There is an obvious mapping from  $(P_G, <_G)$  to its  $\text{ctrl}(-) \cup \text{kind}(-)$ -image which preserves order.

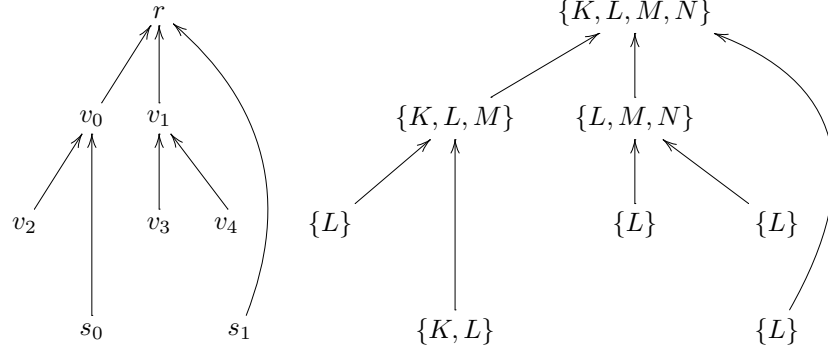


**Figure 6.1:** *The relationship between nesting in a deep kind sorted bigraph and the poset  $(\mathcal{P}(\mathcal{K}), \subseteq)$ .*

There is a good reason why we may wish to consider deep kind sorting instead of (shallow) kind sorting – the sorting of interfaces is very expressive. The sort of a root in a ground bigraph tells us exactly which controls nodes in the bigraph have. The sort of a site in a context (non-ground bigraph) tells us which controls will not be present in any root planted in the site by composition. In terms of spatial logics, if a site  $s$  of a bigraph has no siblings which are sites then it specifies “nowhere under here can we have a node with a control in  $\mathcal{K} \setminus \text{kind}(s)$ .” Similarly, in deep kind sorted parametric reaction rules we can specify the absence of controls anywhere in certain parameters based on the sorting of sites.

The sorting also allows us to identify the absence of certain redexes in certain systems. For example, let  $(R, R')$  be a prime sorted reaction rule with outer sort  $S$ . Let  $G$  be a sorted bigraph with inner width one and sort  $T$  such that  $T \subset S$ . Therefore, no composition  $G \circ F$  can contain a redex of the reaction rule.

Therefore, there may be applications where deep kind sorting bests the shallow sortings. We do not investigate properties of deep kind sorting in this dissertation although our intuition is that it is well-behaved (one could check whether it was a decomposable predicate [16])<sup>1</sup>.

### 6.1.2 Deep sorting with capacities

We can also consider deep sorting with respect to capacities by assigning a value to a control  $K$  specifying how many nodes may lie under a  $K$ -node in a ground bigraph.

**Definition 6.5** (signature with deep capacities). *A signature over  $\mathcal{K}$  with deep capacities is*

<sup>1</sup>Thanks to Lars Birkedal for this suggestion.

composed of a set  $\mathcal{K}$  of controls and three maps:

$$\begin{aligned} \text{arity} : \mathcal{K} &\rightarrow \mathbb{N} & \text{cpc} : \mathcal{K} &\rightarrow \mathbb{N}^\odot \\ \text{status} : \mathcal{K} &\rightarrow \{\text{atomic}, \text{passive}, \text{active}\} \end{aligned}$$

such that if  $\text{status}(K) = \text{atomic}$  then  $\text{cpc}(K) = 0$ .

We consider the sorting where  $\text{cpc}$  defines an exact specification rather than a maximum.

**Definition 6.6** (sorting with deep rigid capacities). *A place-sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  over a signature over  $\mathcal{K}$  with deep capacities is a sorting with deep rigid capacities if  $\Theta = \mathbb{N}^\odot$  and  $\Phi$  requires for all bigraphs  $G$  that:*

**DC1**  $\text{cpc}(p) = p_G + \sum_{s <_G p} \text{cpc}(s)$ , if  $\text{cpc}(p) \neq \odot$ ;

**DC2** if  $\text{cpc}(v) = 0$ ,  $v$  parents no sites;

where  $p$  is a root or node,  $v$  is a node,  $s$  is a site,  $\text{cpc}(s)$  denotes the sort of  $s$ , and  $p_G$  is the number of nodes under  $p$  in  $G$ .

If we remove the sort 0 from  $\Theta$  then we obtain something similar to hard bigraphs and condition DC2 becomes redundant.

It may be possible to combine this sorting with the fundamental kind sorting although we may find the need to impose a tree-structure on the containment graph *i.e.* if a  $K$ -node can contain an  $L$ -node, then we should require that  $\text{cpc}(K) > \text{cpc}(L)$ . We would also require that  $K$ -nodes cannot contain  $K$ -nodes unless their capacity is  $\odot$ .

## 6.2 Link sortings

In this section, we repeat the definition of link sorting and then provide some examples which will be used in the final part of the dissertation.

The first sorting is inspired by modelling tile-based games in bigraphs.<sup>2</sup> It allows us to sort links with directions so that, for example, a *north* port is connected to a *south* port and also allows an ‘unsorted’ sort which corresponds to the usual unconstrained notion of link.

The second sorting has also been called *plain sorting* by Milner. We use it to model typed variants of  $\Lambda\text{BIG}$  in Chapter 11.

<sup>2</sup>Our idea for this came from a talk by Hartmut Ehrig where the double-pushout approach was demonstrated using Pac-Man®.<sup>3</sup> See Ehrig and Padberg’s paper [54].

<sup>3</sup>Pac-Man® NAMCO BANDAI Games Inc.



### 6.2.1 Definitions

Link sorting was introduced by Leifer and Milner and used to present bigraphical models of arithmetic nets and condition-event Petri nets [92]. We reproduce the definitions below. They are quite similar to place sortings. In a sorted signature, sorts are assigned to each port of a control. The interfaces are extended with sorts for names. In link sorted bigraphs, ports (resp. names) inherit their sorts from their control (resp. interface) and a sorting discipline disallows badly sorted bigraphs.

**Definition 6.7** (sorted link graphs). *A signature  $\mathcal{K}$  is  $\Theta$ -sorted if it is enriched by an assignment of a sort  $\theta \in \Theta$  to each  $i \in \text{ar}(K)$  for each control  $K$ . An interface  $X$  is  $\Theta$ -sorted if it is enriched by ascribing a sort to each name  $x \in X$ .*

*A link graph is  $\Theta$ -sorted over  $\mathcal{K}$  if its interfaces are  $\Theta$ -sorted, and for each  $K, i$  the sort assigned by  $\mathcal{K}$  to  $i \in \text{ar}(K)$  is ascribed to the  $i^{\text{th}}$  port of every  $K$ -node.*

$\mathcal{LIG}(\Theta, \mathcal{K})$  denotes the monoidal precategory of sorted link graphs whose identities, composition and tensor product are defined in the obvious way in terms of the underlying (unsorted) link graphs.

**Definition 6.8** (link sorting [92]). *A (link)-sorting (discipline) is a triple*

$$\Sigma = (\mathcal{K}, \Theta, \Phi)$$

*where  $\mathcal{K}$  is  $\Theta$ -sorted, and  $\Phi$  is a condition on  $\Theta$ -sorted link graphs over  $\mathcal{K}$ . The condition  $\Phi$  must be satisfied by the identities and preserved by both composition and tensor product.*

*A link graph in  $\mathcal{LIG}(\Theta, \mathcal{K})$  is said to be  $\Sigma$ -sorted if it satisfies  $\Phi$ . The  $\Sigma$ -sorted link graphs form a monoidal sub-precategory of  $\mathcal{LIG}(\Theta, \mathcal{K})$  denoted by  $\mathcal{LIG}(\Sigma)$ . Further, if  $\mathcal{R}$  is a set of  $\Sigma$ -sorted reaction rules then  $\mathcal{LIG}(\Sigma, \mathcal{R})$  is a  $\Sigma$ -sorted LRS.*

Associated with a link sorting is a sorting functor  $\mathcal{U} : \mathcal{LIG}(\Sigma) \rightarrow \mathcal{LIG}(\mathcal{K})$  which discards sorts. Link sortings which create RPOs and reflect pushouts have desirable properties.

**Theorem 6.9** (useful sortings [92]). *Let  $\mathcal{R}$  be a set of  $\Sigma$ -sorted reaction rules. Then:*

1. *If  $\Sigma$  creates RPOs then bisimilarity for the standard transition system ST over  $\mathcal{LIG}(\Sigma, \mathcal{R})$  is a congruence;*
2. *If in addition  $\Sigma$  reflects pushouts and  $\mathcal{R}$  is simple, then the engaged transitions are adequate for ST.*

**Notation.** *If a name or port  $p$  is assigned a sort  $\theta$  in a bigraph sorted under some link sorting then we denote this by writing  $p : \theta$ .*

### 6.2.2 Tiled link sorting

We now define a link sorting which we call *tiled link sorting* as we will use it to build kind Brss with a ‘tile’ control to model tile-based games.

We presuppose a set  $\text{Dir}$  of *directions*, a disjoint set  $\text{ODir}$  of *opposite directions*, and a bijective map  $op : \text{Dir} \rightarrow \text{ODir}$ . We let  $d$  range over  $\text{Dir}$  and  $o$  range over  $\text{ODir}$ . If  $op(d) = o$  then we say that  $d$  and  $o$  are each other’s *opposite*. Our set of link sorts will be  $\Theta = \text{Dir} \uplus \text{ODir} \uplus \{a\}$  where  $a$  represents an *undirected sort*. If a port, inner name, or open link is assigned a sort  $d$  or  $o$  then we call it *directed*.

**Example 6.10** (cardinal directions). *We define link sorts for the cardinal directions as follows. Let  $\text{Dir} = \{n, e\}$  for ‘north’ and ‘east’ and  $\text{ODir} = \{s, w\}$  for ‘south’ and ‘west’. Define the map  $op$  as  $op(n) = s$  and  $op(e) = w$ .*

**Definition 6.11** (tiled link sorting). *A tiled link sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  is a link sorting where the set of sorts is  $\Theta = \text{Dir} \uplus \text{ODir} \uplus \{a\}$  for arbitrary sets  $\text{Dir}$  and  $\text{ODir}$  with a bijection  $op : \text{Dir} \rightarrow \text{ODir}$  and the condition  $\Phi$  is as follows:*

- *a closed link has either:*
  - *exactly one directed point;*
  - *exactly two directed points with opposite directions;*
  - *arbitrarily many  $a$  points;*
- *an open directed link has exactly one point of equal direction;*
- *an open  $a$ -link has arbitrarily many  $a$ -points.*

The intuition behind the sorting is that the directions are used to sort certain ports so the signature can specify rooms with *e.g.* north, south, east, and west ports which can each be linked uniquely to some room in the obvious manner. We allow edges to have one directed point so that we can represent rooms where there is no exit in that direction *e.g.* no exit to the north. The last sort, the undirected sort  $a$ , is used to represent pure links and so these points and names can be linked together arbitrarily but may not connect to directions. Pure link graphs are recovered (up to categorical equivalence) when  $\text{Dir} = \text{ODir} = \emptyset$ .

The sorting condition is satisfied by the identities and preserved by tensor product.

**Proposition 6.12** (composition respects tiled link sorting). *If  $A : \langle m_0, \theta_0, X_0 \rangle \rightarrow \langle m_1, \theta_1, X_1 \rangle$  and  $B : \langle m_1, \theta_1, X_1 \rangle \rightarrow \langle m_2, \theta_2, X_2 \rangle$  are sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.*

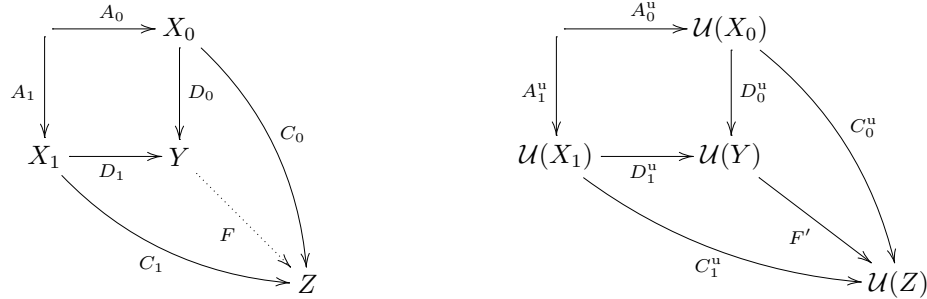
*Proof.* See Appendix A.6, Proposition A.26.  $\square$

Not all arrows in a tiled link-sorted s-category are opcartesian. For example, let  $D = \text{id}_{x:\mathbf{d}}$ ,  $A = \text{id}_{x:\mathbf{d}} \oplus y : \mathbf{a}$  and  $B = (x/x, y)$ .  $B \circ \mathcal{U}(A) = \mathcal{U}(D)$  with  $A$  and  $D$  sorted but  $B$  cannot be sorted with  $x : \mathbf{d}$  as the directed name  $x$  would link two points (*i.e.* tiled link sorting is not a decomposable predicate). Therefore, pushout reflection does not immediately follow. However, we will show in the remainder of this section that a tiled link sorting functor both creates RPOs and reflect pushouts. The following lemma is proven in previous work [118]. The proof given there was lengthy so we will not repeat it.

**Lemma 6.13.** *Let  $\vec{D} : \vec{X} \rightarrow Y$  and  $\vec{C} : \vec{X} \rightarrow Z$  be two bounds for  $\vec{A} : \vec{X} \rightarrow \vec{X}$  in  $\mathcal{LIG}(\Sigma)$  such that each  $y \in Y$  is not idle in at least one of  $(D_0, D_1)$ .*

*If there exists a mediator  $F' : \mathcal{U}(Y) \rightarrow \mathcal{U}(Z)$  such that the diagram on the right commutes, then there exists an  $F : Y \rightarrow Z$  such that  $\mathcal{U}(F) = F'$  and the diagram on the left commutes.*

*Further, if  $F'$  is a unique mediating arrow between  $\vec{D}$  and  $\vec{C}$  then  $F$  is also a unique mediating arrow between  $\vec{D}$  and  $\vec{C}$ .*



We now show that tiled link sorting creates RPOs and reflects pushouts. In the following, we annotate some points with directions or  $\mathbf{a}$  to imply their sort and write  $\text{ts}(G)$  to denote that a bigraph  $G$  obeys tiled link sorting. Note that the proofs do not rely on the cardinality of  $\text{Dir}$  or  $\text{ODir}$ .

**Construction 6.14** (building a tiled link-sorted RPO).

Let  $\vec{A} : H \rightarrow \vec{I}$  have a bound  $\vec{D} : \vec{I} \rightarrow K$ . The sorted RPO is defined as taking the underlying pure RPO  $(\vec{B}' : \vec{I}^u \rightarrow J, B' : J \rightarrow K^u)$  and assigning a sort to the names  $\hat{X}$  of  $J = \langle m, \hat{X} \rangle$ .

The sorting is defined as follows: if  $x$  maps to  $\hat{x}$  in  $B'_0$  or  $B'_1$  then  $\text{sort}(\hat{x}) = \text{sort}(x)$ . This is well-defined and justified in detail in Construction A.27 in Appendix A.6 using Lemma 6.13.  $\square$

**Proposition 6.15.** *A tiled link-sorted RPO for  $\vec{A}$  to  $\vec{D}$  is provided by Construction 6.14.*

*Proof.* This is proven using the faithfulness of the sorting functor and Lemma 6.13.  $\square$

$\mathcal{U}$  creates RPOs. It can also easily be shown that  $\mathcal{U}$  preserves RPOs.

**Proposition 6.16** (tilted link sorting reflects pushouts). *Whenever  $\vec{D}$  bounds  $\vec{A}$  in  $\mathcal{LIG}(\Sigma)$  and  $\mathcal{U}(\vec{D})$  is a pushout for  $\mathcal{U}(\vec{A})$ , then  $\vec{D}$  is a pushout for  $\vec{A}$ .*

*Proof.* This is proven using the faithfulness of the sorting functor, Lemma 6.13, and the fact that an RPO interface (and hence a pushout interface) contains no name which is idle in both legs of the RPO. Alternatively, we could invoke Proposition 5.17 with Lemma 6.13.  $\square$

**Proposition 6.17** (tilted link sorting is safe). *Given a tiled link sorting  $\Sigma$ ,*

1. *Bisimilarity for the standard transition system ST over  $\mathcal{LIG}(\Sigma, \mathcal{R})$  is a congruence.*
2. *In addition, if  $\mathcal{R}$  is simple then the engaged transitions are adequate for ST.*

*Proof.* Follows from Leifer and Milner’s work [92, Theorem 8.6] and Propositions 6.15 and 6.16.  $\square$

We use the following tiled sorting in some of our examples.

**Definition 6.18** (square tile sorting). *Square tile sorting is the tiled link sorting where the link sorts are the cardinal directions i.e.  $\text{Dir} = \{\text{n}, \text{e}\}$  and  $\text{ODir} = \{\text{s}, \text{w}\}$ .*

Although we name these sortings after tile-based board games, they are similar to a generalised form of the directed linear link graphs described by Leifer and Milner once we forget about the undirected sort. Perhaps an intuitive description of tiled link sortings would be as a hybrid of such a generalised directed linear sorting and pure bigraphs (with undirected links).

### 6.2.3 Plain sorting

This link sorting is used as part of our sorting for  $\Lambda\text{BIG}$ . However, it is more generally useful so we present it separately here. We previously called this simply typed link sorting [123] because of its application in that work but we adopt Milner’s terminology here [114].

**Definition 6.19** (plain sorting). *In a plain sorting  $\Sigma = (\mathcal{K}, \Theta, \Phi)$ ,  $\Theta$  is some arbitrary set of types and  $\mathcal{K}$  is a  $\Theta$ -sorted binding signature. The condition  $\Phi$  is:*

- *if  $\text{link}(p) = x$  then  $\text{sort}(p) = \text{sort}(x)$  for points  $p$  and outer name  $x$ , and*
- *edges link points and inner names of the same sort.*

We write  $\text{type}(q) = \text{sort}(q)$  where  $q$  is a port or a name.

The sorting has the simple condition that points are connected to links with the same type. It can be applied to pure or local bigraphs (in the latter, replace ‘edges’ with ‘binding ports’ in the definition). It is satisfied by the identities and preserved by composition and tensor product. We denote the s-category of plain sorted local bigraphs over a signature  $\mathcal{K}$  as  $\text{SBG}_{\text{loc}}(\mathcal{K}^{\text{ps}})$ . The forgetful functors

$$\text{SBG}_{\text{loc}}(\mathcal{K}^{\text{ps}}) \xrightarrow{\mathcal{U}_{\text{type}}} \text{BIG}_{\text{loc}}(\mathcal{K}) \xrightarrow{\mathcal{U}_{\text{loc}}} \text{BIG}(\mathcal{K}^{\text{u}})$$

respectively forget sorting and locality of interfaces ( $\text{BIG}_{\text{loc}}$  is the s-category of local bigraphs). See Milner’s work for a definition of  $\mathcal{U}_{\text{loc}}$  [112].

**Notation.** A local interface sorted over  $\Sigma = (\mathcal{K}, \Theta, \Phi)$  is written as  $I = \langle m, X, \text{loc}, \text{type}_I \rangle$  where  $\text{type} : X \rightarrow \Theta$  assigns types to names.

The underlying local bigraph of a sorted local bigraph  $G$  is defined as  $\mathcal{U}_{\text{type}}(G)$ . The underlying pure bigraph of a sorted local bigraph  $G$  is defined as  $(\mathcal{U}_{\text{loc}} \circ \mathcal{U}_{\text{type}})(G)$  and denoted  $G^{\text{u}}$ .

The parallel product of two sorted local interfaces is defined when the union of their type maps is single-valued. The parallel product of two sorted local bigraphs  $G$  and  $G'$  is defined when the parallel product of their interfaces and  $\mathcal{U}_{\text{type}}(G) \parallel \mathcal{U}_{\text{type}}(G')$  are. Essentially, parallel product respects typing.

Pure or local bigraphs under this sorting are not quite opcartesian (Milner leaves this as an exercise for the reader [114]) however the sorting is preserved by decomposition when the inner bigraph has no idle names, a property which is stronger than sufficiently jointly opcartesian.

**Lemma 6.20.** Given two plain sorted bigraphs  $D : H \rightarrow J$  and  $A : H \rightarrow I$  where  $A$  has no idle links, let  $B' : \mathcal{U}_{\text{type}}(I) \rightarrow \mathcal{U}_{\text{type}}(J)$  be a local bigraph such that  $\mathcal{U}_{\text{type}}(D) = B' \circ \mathcal{U}_{\text{type}}(A)$ . Then there exists a unique sorted bigraph  $B : I \rightarrow J$  such that  $\mathcal{U}_{\text{type}}(B) = B'$  and  $D = B \circ A$ .

*Proof.* The situation is depicted below.

$$\begin{array}{ccc}
 \begin{array}{c} \langle n, Y, \text{loc}_J, \text{type}_J \rangle \\ \uparrow B \\ \langle m, X, \text{loc}_I, \text{type}_I \rangle \\ \uparrow A \\ \langle h, W, \text{loc}_H, \text{type}_H \rangle \end{array} & \xrightarrow{\mathcal{U}_{\text{type}}} & \begin{array}{c} \langle n, Y, \text{loc}_J \rangle \\ \uparrow B' \\ \langle m, X, \text{loc}_I \rangle \\ \uparrow \mathcal{U}_{\text{type}}(A) \\ \langle h, W, \text{loc}_H \rangle \end{array} \\
 D \curvearrowright & & \curvearrowright \mathcal{U}_{\text{type}}(D)
 \end{array}$$

Define  $B$  as  $B'$  with inner interface  $I$  and outer interface  $J$ . We will prove that  $B$  is well-sorted.

Let  $x \in X$  and  $\text{link}_B(x) = l$  for some link  $l$  of  $B$ .  $x$  is not idle in  $A$  so there exists a point  $p$  in  $A$  such that  $\text{link}_A(p) = x$  and so  $\text{link}_D(p) = l$ . As  $A$  and  $D$  are sorted, we conclude that  $\text{type}(p) = \text{type}(x) = \text{type}(l)$ . Therefore,  $B$  is well-sorted on inner names.

Let  $p$  be a port of  $B$  and  $\text{link}_B(p) = l$  for some link  $l$  of  $B$ . Then  $\text{link}_D(p) = l$  and  $\text{type}(p) = \text{type}(l)$ . Therefore,  $B$  is also well-sorted on ports and hence well-sorted.

We have  $B : I \rightarrow J$  such that  $\mathcal{U}_{\text{type}}(B) = B'$ . As  $\mathcal{U}_{\text{type}}$  is faithful,  $B$  is unique.  $\square$

**Construction 6.21** (building a plain-sorted RPO).

Let  $\vec{A} : H \rightarrow \vec{I}$  have a bound  $\vec{D} : \vec{I} \rightarrow J$  in a s-category of sorted bigraphs with  $H = \langle h, W, \text{loc}_H, \text{type}_H \rangle$ ,  $I_i = \langle m_i, X_i, \text{loc}_i, \text{type}_i \rangle$ , and  $J = \langle n, Y, \text{loc}_J, \text{type}_J \rangle$ . We construct an RPO  $(\vec{B}, B)$  for  $\vec{A}$  to  $\vec{D}$  as follows.

First build a local RPO  $(\vec{B}_{\text{loc}}, B_{\text{loc}})$  for  $\mathcal{U}_{\text{type}}(\vec{A})$  to  $\mathcal{U}_{\text{type}}(\vec{D})$  where the inner face of  $B$  is  $\langle m, X, \text{loc}_I \rangle$ .

The set  $X$  is defined as in the pure construction (Construction 2.38) for link graph RPOs as follows. Let  $V_i$  be the nodes of  $A_i$ ,  $V_2 = V_0 \cap V_1$ , and  $V_i - V_2 \uplus V_3$  be the nodes of  $D_i$ . Edges  $E_i$  (recovered from  $\mathcal{U}_{\text{loc}} \circ \mathcal{U}_{\text{type}}$ ) are treated similarly and ports  $P_i$  are treated like their nodes. First, define:

$$X'_i \stackrel{\text{def}}{=} \{x \in X_i \mid D_i(x) \in E_3 \uplus Y\}.$$

Next, define  $\cong$  to be the smallest equivalence on the disjoint sum  $X'_0 + X'_1$  for which

$$(0, x_0) \cong (1, x_1) \text{ whenever } A_0(p) = x_0 \text{ and } A_1(p) = x_1 \text{ for some } p \in W \uplus P_2$$

and define  $X \stackrel{\text{def}}{=} (X'_0 + X'_1) / \cong$ . For each  $x \in X'_i$  the  $\cong$ -equivalence class of  $(i, x)$  is denoted by  $\widehat{i, x}$ .

We define the function  $\text{type}_I$  as  $\text{type}(\widehat{0, x_0}) = \text{type}(x_0)$ .  $\text{type}_I$  is well-defined as follows. If  $(0, x_0) \cong (1, x_1)$  then  $A_0(p) = x_0$  and  $A_1(p) = x_1$ . Since  $A_0$  and  $A_1$  are sorted,  $\text{type}(p) = \text{type}(x_0) = \text{type}(x_1)$ . This implies that all  $\cong$ -equivalent names  $x_i$  have the same type. The sorted RPO is then defined as  $(\vec{B}_{\text{loc}}, B_{\text{loc}})$  lifted to the sorted setting and with the inner interface of  $B$  defined as  $\langle m, X, \text{loc}_I, \text{type}_I \rangle$ .  $\square$

**Proposition 6.22** (valid RPO construction). *Construction 6.21 builds RPOs in  $\text{SBG}_{\text{loc}}$ .*

*Proof.* See Appendix A.6.2, Proposition A.28.  $\square$

**Proposition 6.23** (plain sorting reflects pushouts). *Whenever  $\vec{D}$  bounds  $\vec{A}$  in  $\mathcal{LIG}(\Sigma)$  and  $\mathcal{U}(\vec{D})$  is a pushout for  $\mathcal{U}(\vec{A})$ , then  $\vec{D}$  is a pushout for  $\vec{A}$ .*

*Proof.* This is proven using the faithfulness of the sorting functor, Lemma 6.20, and the fact that an RPO interface (and hence a pushout interface) contains no name which is idle in both legs of the RPO. See Proposition 5.17.  $\square$

**Proposition 6.24** (plain sorting is safe). *Given a plain sorting  $\Sigma$ ,*

1. *Bisimilarity for the standard transition system  $\text{ST}$  over  $\mathcal{LIG}(\Sigma, \mathcal{R})$  is a congruence.*
2. *In addition, if  $\mathcal{R}$  is simple then the engaged transitions are adequate for  $\text{ST}$ .*

*Proof.* Follows from Leifer and Milner’s work [92, Theorem 8.6] and Propositions 6.22 and 6.23.  $\square$

## 6.3 Compound sortings

Birkedal et al. give examples of how to combine sortings by composition and pullbacks of sorting functors [16]. We present the latter notion slightly differently here to match our notation.

### 6.3.1 Combining sortings

The simplest way to combine sortings is to compose them. This is a useful way to restrict a sorting or to layer one sorting over another. We have already demonstrated the former idea in Chapter 4 by considering subcategories of sorted s-categories. We will present an example of the latter in Section 11.2.2.

**Proposition 6.25.** *The composition of two sorting functors  $\mathcal{G} : \mathbf{E} \rightarrow \mathbf{D}$  and  $\mathcal{F} : \mathbf{D} \rightarrow \mathbf{C}$  is a sorting functor.*

As Birkedal et al. note, the composition of sorting functors preserves RPO creation. Composition also preserves pushout reflection *i.e.* if  $\mathcal{G}$  and  $\mathcal{F}$  reflect pushouts then  $\mathcal{G} \circ \mathcal{F}$  does. They also consider combining sortings via pullback.

**Proposition 6.26** ([16]). *The pullback of functors  $\mathcal{D} : \mathbf{D} \rightarrow \mathbf{B}$  and  $\mathcal{E} : \mathbf{E} \rightarrow \mathbf{B}$  is isomorphic to the category  $\mathbf{D} \times_{\mathbf{B}} \mathbf{E}$ , which has objects  $\{(A, B) \mid \mathcal{D}(A) = \mathcal{E}(B)\}$  and morphisms  $\{(f, g) \mid \mathcal{D}(f) = \mathcal{E}(g)\}$ .*

Moreover, if  $\mathcal{D}$  and  $\mathcal{E}$  are sortings, then so are  $\pi_{\mathbf{D}}$  and  $\pi_{\mathbf{E}}$ .

$$\begin{array}{ccc} \mathbf{D} \times_{\mathbf{B}} \mathbf{E} & \xrightarrow{\pi_{\mathbf{E}}} & \mathbf{E} \\ \pi_{\mathbf{D}} \downarrow & & \downarrow \mathcal{E} \\ \mathbf{D} & \xrightarrow{\mathcal{D}} & \mathbf{B} \end{array}$$

We consider a special case of combining sortings of bigraphs via pullback where the signatures of the sorted s-categories are compatible *i.e.* both sortings extend the same signature by sharing the same set of controls and agreeing on common mappings. This restriction is unnecessary but reflects how we will combine sortings in the dissertation and allows us to introduce a more succinct notation.

We usually represent sorts of sorted interfaces as vectors but to ease the definition below, we consider the extra structure a sorting may add to interfaces as a sequence of *functions* *e.g.* a fundamental kind sorting adds the function  $kind : m \rightarrow \mathcal{P}(\mathcal{K})$  to a pure interface  $\langle m, X \rangle$ . In the context of this section of the dissertation,  $\wedge$  denotes logical conjunction.

**Definition 6.27** (paired sorting). *Given two sortings  $\Sigma_1 = (\mathcal{K}_1, \Theta_1, \Phi_1)$  and  $\Sigma_2 = (\mathcal{K}_2, \Theta_2, \Phi_2)$  over a compatible signature  $\mathcal{K}$ , the paired sorting is the sorting  $\Sigma_1 \times_{\mathcal{K}} \Sigma_2 = (\mathcal{K}, \Theta_1 \times \Theta_2, \Phi_1 \wedge \Phi_2)$  on  $\Theta_1 \times \Theta_2$ -sorted pure/local bigraphs over  $\mathcal{K}$ .*

*The condition  $(\Phi_1 \wedge \Phi_2)(G)$  on  $G : \langle m, X, f_1, \dots, f_i, g_1, \dots, g_j \rangle \rightarrow \langle n, Y, f'_1, \dots, f'_{i'}, g'_1, \dots, g'_{j'} \rangle \in \mathcal{BIG}(\Sigma_1 \times_{\mathcal{K}} \Sigma_2)$  is true exactly when for any pair of arrows*

$$G_1 : \langle m, X, f_1, \dots, f_i \rangle \rightarrow \langle n, Y, f'_1, \dots, f'_{i'} \rangle \in \mathcal{BIG}(\Sigma_1),$$

$$G_2 : \langle m, X, g_1, \dots, g_j \rangle \rightarrow \langle n, Y, g'_1, \dots, g'_{j'} \rangle \in \mathcal{BIG}(\Sigma_2)$$

*with the same underlying pure/local bigraph,  $\Phi_1(G_1) \wedge \Phi_2(G_2)$ . The (projection) sorting functors  $\mathcal{U}_{\Sigma_1} : \Sigma_1 \times_{\mathcal{K}} \Sigma_2 \rightarrow \Sigma_1$  and  $\mathcal{U}_{\Sigma_2} : \Sigma_1 \times_{\mathcal{K}} \Sigma_2 \rightarrow \Sigma_2$  each forget one of the sortings.*

This is isomorphic to a special case of the pullback construction.

**Lemma 6.28.**  $(\Phi_1 \wedge \Phi_2)(G) \equiv \Phi_1(\mathcal{U}_{\Sigma_2}(G)) \wedge \Phi_2(\mathcal{U}_{\Sigma_1}(G)).$

*Proof.* By Definition 6.27. □

**Proposition 6.29.** *If the functors  $\mathcal{U}_i : \mathcal{BIG}(\Sigma_i) \rightarrow \mathcal{BIG}(\mathcal{K})$  create RPOs then so does  $\mathcal{U}_i \circ \mathcal{U}_{\Sigma_i}$ .*

*Proof.* See Appendix A.6, Proposition A.29. □



**Proposition 6.30.** *If the functors  $\mathcal{U}_i : \mathcal{BIG}(\Sigma_i) \rightarrow \mathcal{BIG}(\mathcal{K})$  (weakly) reflect pushouts then so does  $\mathcal{U}_i \circ \mathcal{U}_{\Sigma_i}$ .*

*Proof.* See Appendix A.6, Proposition A.30. □

As conjunction is associative, the definition of paired sorting (and pullback of functors) and the proofs of RPO creation and pushout reflection generalise to combinations of multiple sortings. This is unsurprising since the two sortings are treated somewhat separately.

### 6.3.2 Kind rigid control-sorting with visibility

In order to demonstrate a paired sorting, we will combine a kind sorting with visibility with the rigid control-sorting of Birkedal et al. [15] who propose an instance of the latter as a means of modelling context-aware systems. We use this combination in Section 10.5 to present a variation of one of their bigraphical algorithms.

**Definition 6.31** (Rigid control-sorting [15]). *A place-sorting  $\Sigma_K = (\mathcal{K}, \Theta, \Phi)$  is a rigid control-sorting if it is equipped with a predicate  $\phi$  and*

- $\Theta \subseteq \mathcal{P}(\mathcal{K})$ .

*Then  $\Phi$  requires for all bigraphs  $G$  that:*

**P1** *if  $r = G^*(s)$  then  $\text{sort}(s) = \text{sort}(r)$ ;*

**P2** *if  $r = G^*(v)$  then  $\phi(\text{ctrl}_G(v), \text{sort}(r))$ ;*

*where  $r$  is a root,  $s$  a site,  $v$  is a node, and  $G^*$  is the function which takes each site or node of  $G$  to the unique root above it.*

**Definition 6.32** (kind rigid control-sorting with visibility). *A place-sorting  $\Sigma_K = (\mathcal{K}, \Theta, \Phi)$  is a kind rigid control-sorting with visibility if it is defined over a kind signature with visibility and with a predicate  $\phi$  and*

- $\Theta = \mathcal{P}(\mathcal{K}_{\text{vis}}) \times \Theta_2$  where  $\Theta_2 \subseteq \mathcal{P}(\mathcal{K})$ .

*Then  $\Phi$  requires for all bigraphs  $G$  that:*

**KV1** *if  $p = G(v)$  then  $\text{ctrl}(v) \in \text{sort}_1(p)$ ;*

**KV2** *if  $p = G(s)$  then  $\text{sort}_1(s) \subseteq \text{sort}_1(p)$ ;*

**KV3** if  $K \in \text{kind}_{\text{inv}}(v)$  then  $v_{K,G} = 1$ ;

**KV4** if  $\text{sort}_1(v) = \emptyset$ ,  $v$  has no children;

**P1** if  $r = G^*(s)$  then  $\text{sort}_2(s) = \text{sort}_2(r)$ ;

**P2** if  $r = G^*(v)$  then  $\phi(\text{ctrl}_G(v), \text{sort}_2(r))$ ;

where  $p$  is a root or node,  $\text{sort}_1(v)$  is the sort of a node,  $r$  is a root,  $s$  a site,  $v$  is a node, and  $G^*$  is the function which takes each site or node of  $G$  to the unique root above it.

**Corollary 6.33.** *Kind rigid control-sorting with visibility creates RPOs.*

*Proof.* By Corollary 5.30, RPO creation of rigid control-sorting [15], and Proposition 6.29.  $\square$

Kind rigid control-sorting with visibility is a paired place-sorting and generalises both of its components (up to categorical equivalence). The kind sorting is the special case where  $|\Theta_2| = 1$  and  $\phi$  is vacuously true. Rigid control-sorting is the special case where all controls are visible, each non-atomic control has kind sort  $\mathcal{K}$ , and each atomic control has sort  $\emptyset^4$ .

It is important to consider which sortings to combine. The pointwise nature of our combination allows us to combine sortings with *conflicting* conditions. This may remove bigraphs we wish to keep.

We finally note that the combination of kind sorting and rigid control-sorting is not ideal due to this pointwise combination. It would make sense to add another rule to  $\Phi$

$$\text{KVP1} \quad \text{sort}_1(t) \subseteq \text{sort}_2(t)$$

where  $t$  is a site or root of the bigraph  $G$ . This would eliminate some arrows of the s-category whose kind interface sort has some useless elements. It forms a subcategory of our combination. We have not taken this approach here as we have seen that forming subcategories can break RPO creation. It is likely that this extra rule is well-behaved but we leave the problem open.

---

<sup>4</sup>This retrieval of rigid control-sorting is suboptimal as many bigraphs with different kind interface sorts correspond to one rigid control-sorted bigraph. If the full subcategory of a kind s-category with the one interface sort  $\mathcal{K}$  creates RPOs – and it probably does in this case – this would yield a better match.

## 6.4 Summary

This chapter presented variations of kind sortings, useful link sortings, and discussed existing methods of combining sortings.

It was demonstrated in this chapter (and in Appendix A.6) that plain-sorted bigraphs are not opcartesian but are sufficiently jointly opcartesian with respect to their sorting functors. This is also true of the  $\sigma\lambda\nu\delta$ -sorting discussed in Section 11.2. This property facilitated simple proofs of pushout reflection based on a knowledge of pushouts in the respective underlying s-categories.

## 6.5 Related work

Many sorting disciplines and generalisations have been introduced for pure bigraphs at this point. For linking; binding bigraphs [74, 73] and local bigraphs [106] respectively admit name-binding and name-scoping, Leifer and Milner consider directed links [92], and Bundgaard and Sassone introduced subsorting [24]. Grohmann and Miculan’s *directed bigraphs* generalise link graphs significantly, taking ideas from Sassone and Sobociński [138]. Directed bigraphs appear to lack a notion of name-scoping although this could be added.

We include activity in the definition of dynamic signature as in previous work [74]. Jensen considers a yet simpler signature without activity as basic and then provides activity on top of this signature by means of a safe place sorting, yielding a more elegant treatment of the theory [73].

There are strong similarities between kind sorting and the definition of sorting for the polyadic  $\pi$ -calculus [101]. We use the set  $\mathcal{K}$  as the set of ‘subject sorts’ and subsets of  $\mathcal{K}$  as the set of ‘object sorts.’ In the polyadic  $\pi$ -calculus, the set of objects sorts is the set of sequences of subject sorts. Kind sorting with rigid capacities is closest to the polyadic  $\pi$ -calculus approach as it specifies exact numbers of child controls. However, the only notion of ordering children it has is provided by multi-nodes and as our current definition of multi-nodes is shallow (invisible controls cannot parent invisible controls), these cannot represent nested sequences.

Less work has been done on place-sorting, presumably as linking is powerful enough to allow encodings of many calculi although without place-sorting, not all terms in the Brs will correspond to terms of the calculus. Milner introduced the general definition of place sorting when presenting a homomorphic sorting to sort finite CCS. The sorting allowed a proof that *mono bisimilarity* in the Brs coincides with bisimilarity in CCS. Kind sortings are a particular example of Milner’s definition of place sorting which generalise homomorphic sortings (see Appendix A.2).

Birkedal et al. have proposed a method of modelling context-aware systems by describing

a single bigraphical system as a combination of three smaller component systems where each component represents a different view of the world and agents interact with a context via a shared proxy [15, 14]. To ensure compositional safety, they use a particular example of their safe rigid control-sortings.

Their definition of rigid control-sorting has minor similarities with deep kind sorting which was proposed earlier [118]. It differs from fundamental kind sorting in that the containment relationship is between a root and the nodes below the root rather than the parent-child relationship of kind sortings. For this reason, neither sorting generalises the other except in trivial cases. However, we saw in Section 6.3.2 that they are easily combined safely. We demonstrate a use for this combination in Chapter 10.5.

Birkedal, Debois, and Hildebrandt [16] have introduced *predicate sortings*, a general theory of safe sorting for reactive systems over categories which will hopefully be extended to cover precategories and 2-categories – bigraphs are defined in the former setting and the latter has been studied as an alternative setting for reactive systems [137]. Predicate sortings subsume sortings where the sorting condition is retained by composition/decomposition of bigraphs. The relationship between kind sortings and predicate sortings *i.e.* can kind sortings be couched as predicate sortings, deserves attention. So too does the expressiveness of their sorted reaction relations with respect to guaranteed *absence*, a notion we explore in Chapter 10 for kind sortings. As mentioned in the last chapter, the conditions they require of sorting functors to imply safety do not hold in general for kind sortings with invisible controls or for certain safe subcategories thereof but do hold when no invisible controls are present in the signature.

## Summary of Part I

We introduced our enrichments of pure bigraphs. We satisfied our hypothesis by proving that these enrichments and some of their subcategories retain the dynamic theory of pure bigraphs.

We leave kind sortings (and indeed bigraphs) in the next part where we investigate Milner's explicit substitution calculus.

## Part II

# Milner's $\Lambda_{\text{sub}}$ Calculus

---

## Outline of Part II

This part of the dissertation concerns  $\Lambda_{\text{sub}}$ , Milner's  $\lambda$ -calculus with explicit substitutions.  $\Lambda_{\text{sub}}$  is an atypical explicit substitution (ES) calculus as it has no rules for propagating substitutions through terms. Instead, substitution is performed at a distance or non-locally.

We will prove confluence and normalisation properties of the calculus using traditional methods for the former and introducing a new method for the latter. Our main contribution in this section is the method of simulating non-local substitution of  $\Lambda_{\text{sub}}$  with the local substitution of other calculi. This was motivated by the problems of inducting over a substitution calculus which lacks inductive propagation rules. We first apply the simulation to prove that  $\Lambda_{\text{sub}}$  preserves strong normalisation of  $\beta$ -reduction (PSN) and then apply it to derive a neat characterisation of the strongly normalising terms of the calculus by using an intersection type system.

A secondary contribution is the comparison between Milner's calculus and other ES calculi. We give our explanation of how  $\Lambda_{\text{sub}}$  has both full composition of substitutions whilst retaining the PSN property. Until recently, this combination was an open problem in the area.

Sections 9.1, 9.3, and 9.4 are joint work with Delia Kesner.

# Chapter 7

## $\lambda$ -calculi with Explicit Substitution

*You can't lay a patch by computer design*

*It's just a lot of stupid, stupid signs*

*The Sidewinder Sleeps Tonite – R.E.M.*

### 7.1 Introduction

This chapter introduces the different explicit substitution (ES) calculi under discussion. We first introduce Bloo and Rose's  $\lambda_{\text{xgc}}$  calculus which has a rule for explicit garbage collection. This feature, introduced to ease the proof of preservation of strong normalisation (PSN) for  $\lambda_{\text{xgc}}$ , is present in all the calculi we consider. Having noticed similarities between this calculus and  $\Lambda_{\text{sub}}$ , we use Bloo and Rose's methods to prove confluence for Milner's calculus in Chapter 8.

Kesner and Lengrand's  $\lambda_{\text{lrx}}$  calculus was the first ES calculus with a local propagation calculus to be published which had the properties of confluence, preservation of strong normalisation, step by step simulation of  $\beta$ -reduction, and full composition. It differs from many other named ES calculi in that it is not an extension of the  $\lambda$ -calculus – terms in  $\lambda_{\text{lrx}}$  are linear encodings of  $\lambda$ -terms. This linearity ensures that PSN is retained.

Kesner's  $\lambda_{\text{es}}$  calculus has achieved the same results with a much simplified theory;  $\lambda_{\text{es}}$  extends the  $\lambda$ -calculus. We compare  $\Lambda_{\text{sub}}$  to both calculi. Their similarities are exploited in Chapter 9 to give a proof of PSN and a characterisation of the strongly normalising terms of  $\Lambda_{\text{sub}}$ .



$\Lambda_{\text{sub}}$  was introduced by Milner [107] to present a bigraphical encoding of the  $\lambda$ -calculus (see Section 11.1), demonstrate an application of local bigraphs, and begin the study of confluence in Brss. While it is certainly confluent (as we show later), we demonstrate in this part that it is in fact very close operationally to the pure  $\lambda$ -calculus.

By ‘operationally close’ we mean that it can simulate  $\beta$ -reduction step-by-step, is confluent, preserves strong normalisation of  $\beta$ -reduction, and has full composition of substitutions (see Section 7.3.7). We dedicate this part of the dissertation proving these properties by existing and novel means. Furthermore, we show that its set of strongly normalising terms can be characterised by a simple extension of the intersection type system which characterises the strongly normalising pure terms. It follows from this that every strongly normalising term of  $\Lambda_{\text{sub}}$  is a reduct of a pure strongly normalising term.

Putting all these properties together,  $\Lambda_{\text{sub}}$  (and hence its image in  $\Lambda\text{BIG}$ ) can be seen as a true extension of the  $\lambda$ -calculus with explicit substitutions. We use these results to reason about bigraphical models of typed  $\Lambda_{\text{sub}}$  in the next part of the dissertation.

## 7.2 The calculi under discussion

The various calculi use different notation to represent explicit substitutions. For consistency and space, we mainly adopt Kesner’s notation [78] in the following but we alter the names of reduction and equivalence relations to highlight commonalities.

### 7.2.1 The $\lambda\text{xgc}$ calculus

$\lambda\text{xgc}$  is due to Bloo and Rose [20, 133, 18] and is a refinement of the  $\lambda$ -calculus in the tradition of  $\lambda\sigma$  [1]. It is an explicit substitution calculus with (explicit) garbage collection.

The set  $\Lambda\text{x}$  of  $\lambda\text{xgc}$  terms is inductively defined by

$$t ::= x \mid \lambda x.t \mid t t \mid t[x/u]$$

where the notation  $t[x/u]$  represents an explicit substitution or *closure*. The set of terms for  $\Lambda_{\text{sub}}$  and  $\lambda\text{es}$  is also  $\Lambda\text{x}$ . As stated in the introduction, we will consider all reduction rules to be closed under contexts modulo  $\equiv$ .

**Definition** ( $\lambda\text{xgc}$  reduction).  *$\lambda\text{xgc}$  reduction, or  $\rightarrow_{\text{bxcg}}$ , is defined as the union of the rules in Figure 7.1. We define the relations  $\rightarrow_{\text{x}} = \rightarrow_{\text{Var}} \cup \rightarrow_{\text{Vargc}} \cup \rightarrow_{\text{Lamb}} \cup \rightarrow_{\text{App1}}$  and  $\rightarrow_{\text{gc}} = \rightarrow_{\text{x}} \cup \rightarrow_{\text{gc}}$ .*

|                      |                                  |                                |                           |
|----------------------|----------------------------------|--------------------------------|---------------------------|
| $(\lambda x.t)u$     | $\longrightarrow_b$              | $t[x/u]$                       | (substitution generation) |
| $x[x/t]$             | $\longrightarrow_{\text{Var}}$   | $t$                            | } (local substitution)    |
| $(t_1 t_2)[y/u]$     | $\longrightarrow_{\text{App}_1}$ | $t_1[y/u] t_2[y/u]$            |                           |
| $(\lambda x.t)[y/u]$ | $\longrightarrow_{\text{Lamb}}$  | $\lambda x.t[y/u]$             |                           |
| $x[y/t]$             | $\longrightarrow_{\text{Vargc}}$ | $x$ if $x \neq y$              |                           |
| $t[x/u]$             | $\longrightarrow_{\text{gc}}$    | $t$ if $x \notin \text{FV}(t)$ | (garbage collection)      |

**Figure 7.1:** Reduction rules for  $\lambda\text{xgc}$ 

In the (**Lamb**) rule,  $x \neq y$  is implicit by the variable convention (see Page 35). We name  $\rightarrow_x$  local substitution as its purpose is to distribute an ES  $[y/u]$  through terms via the (**Lamb**) and (**App**<sub>1</sub>) rules and then either replace a variable  $y$  or discard the ES. The substitution only interacts with the part of the term which is directly below it, or *local* to it.

Notice that the rules do not allow one substitution to push inside another *e.g.* the term  $x[y/z][z/t]$  with  $x \neq y \neq z$  has only four  $\rightarrow_{\text{bxgc}}$  sequences, all of which have the same terms:

$$x[y/z][z/t] \rightarrow_{\text{Vargc/gc}} x[z/t] \rightarrow_{\text{Vargc/gc}} x.$$

In other words, there is no composition of substitutions in  $\lambda\text{xgc}$ . Melliès noted that interactions between the propagation rules and composition of substitutions can break PSN [98]. Bloo and Rose avoid this by disallowing composition with the result that  $\lambda\text{xgc}$  is confluent, can simulate  $\beta$ -reduction step-by-step, and preserves strong normalisation of  $\beta$ -reduction [20]. It is not confluent on open terms however.

The explicit garbage collection rule which allows garbage substitutions to be discarded non-locally is an important feature of  $\lambda\text{xgc}$  (due to Rose [131]). The rule eases many of Bloo and Rose's inductive proofs and appears in all the calculi we concentrate on.

### 7.2.2 $\lambda\text{lxr}$

$\lambda\text{lxr}$  [80, 79] is an explicit substitution calculus which is a sound and complete computational counterpart to the intuitionistic part of the Proof Nets of Linear Logic [61]. It is also the first (published) named explicit substitution calculus to our knowledge to enjoy the properties of closed confluence, preservation of strong normalisation, and full composition of substitutions.

$\lambda\text{lxr}$  builds partly on work by David and Guillaume on the  $\lambda_{ws}$  calculus [46]. The  $\lambda_{ws}$  calculus

allowed a level of composition of substitutions whilst retaining PSN and was the first<sup>1</sup> explicit substitution calculus which satisfied step-by-step simulation of  $\beta$ -reduction, confluence on terms with metavariables, and PSN.

Terms in  $\lambda\text{lxr}$  are *linear* (see below); weakenings and contractions are used to allow this. This linearity avoids many problems where composition of substitutions usually break PSN such as the needless copying of an explicit substitution. Fernández and Mackie [55] explored these notions in earlier work.

We concentrate on the properties of full composition and PSN here and refer the reader to the original works for the connections to Linear Logic.

The set of terms of  $\lambda\text{lxr}$  extends  $\Lambda x$  with *explicit weakenings* and *explicit contractions* and is defined as:

$$t ::= x \mid \lambda x.t \mid tt \mid t[x/t] \mid W_x(t) \mid C_x^{y,z}(t).$$

The constructors  $W_x(t)$  and  $C_x^{y,z}(t)$  are explicit weakenings and explicit contraction respectively with free variables defined as  $\text{FV}(W_x(t)) = \text{FV}(t) \cup \{x\}$  and  $\text{FV}(C_x^{y,z}(t)) = (\text{FV}(t) \setminus \{y, z\}) \cup \{x\}$ .

The term  $W_x(t)$  is an annotated form of  $t$  which states that the free variable  $x$  does not occur free in  $t$ . As it is explicitly part of the syntax, it can play a rôle in the reduction relation of  $\lambda\text{lxr}$  and weakenings are in fact used to provide an explicit garbage collection rule. Consider the term  $W_x(t)[x/u]$ . As  $x$  does not occur free in  $t$ , we may want to garbage-collect the substitution. The rule **Weak**<sub>1</sub> in Figure 7.3 does precisely this. Weakenings in  $\lambda\text{lxr}$  may always be pulled out to the top level or their binder, allowing efficient garbage collection.

Substitution in  $\lambda\text{lxr}$  is local and defined with a set of distributive rules. Weakenings also allow efficient propagation of substitutions. For example, propagating the substitution  $[x/u]$  through  $W_x(t)$  is pointless as no substitution can take place and so the reduction rules do not permit this propagation.

Weakenings allow free variables to be kept through reduction. The two destructive rules are **Var** and **Weak**<sub>1</sub>. As expected, the substitution rule **Var** does not lose free variables. Interestingly, the garbage collection rule **Weak**<sub>1</sub> remembers the free variables of the discarded substitution via a weakening. Kesner and Lengrand compare this preservation of free variables to “interface preserving” [86] in interaction nets.

Contractions in  $\lambda\text{lxr}$  allow the linearity of terms discussed below. The term  $C_x^{y,z}(t)$  may be read as ‘ $t$  where  $y$  and  $z$  are  $x$ .’

We consider the *linear* terms of  $\lambda\text{lxr}$ . A linear term is a term in which each variable has a

---

<sup>1</sup>To our knowledge.

|                                |                   |                                |                                                     |
|--------------------------------|-------------------|--------------------------------|-----------------------------------------------------|
| $C_w^{x,v}(C_x^{z,y}(t))$      | $\equiv_A$        | $C_w^{x,y}(C_x^{z,v}(t))$      | if $x \neq y, v$                                    |
| $C_x^{y,z}(t)$                 | $\equiv_{C1_c}$   | $C_x^{z,y}(t)$                 |                                                     |
| $C_{x'}^{y',z'}(C_x^{y,z}(t))$ | $\equiv_{C2_c}$   | $C_x^{y,z}(C_{x'}^{y',z'}(t))$ | if $x \neq y', z' \& x' \neq y, z$                  |
| $W_x(W_y(t))$                  | $\equiv_{C_w}$    | $W_y(W_x(t))$                  |                                                     |
| $t[x/v][y/u]$                  | $\equiv_S$        | $t[y/u][x/v]$                  | if $y \notin \text{FV}(v) \& x \notin \text{FV}(u)$ |
| $C_w^{y,z}(t)[x/v]$            | $\equiv_{Cont_2}$ | $C_w^{y,z}(t[x/v])$            | if $x \neq w \& y, z \notin \text{FV}(v)$           |

**Figure 7.2:** Congruences for  $\lambda\text{lxr}$ 

|                          |                                    |                                           |                                                                                              |
|--------------------------|------------------------------------|-------------------------------------------|----------------------------------------------------------------------------------------------|
| $(\lambda x.t)u$         | $\longrightarrow_b$                | $t[x/u]$                                  |                                                                                              |
| System x                 |                                    |                                           |                                                                                              |
| $x[x/t]$                 | $\longrightarrow_{\text{Var}}$     | $t$                                       |                                                                                              |
| $(t\ u)[x/v]$            | $\longrightarrow_{\text{App}_3}$   | $t[x/v]\ u$                               | $x \in \text{FV}(t)$                                                                         |
| $(t\ u)[x/v]$            | $\longrightarrow_{\text{App}_2}$   | $t\ u[x/v]$                               | $x \in \text{FV}(u)$                                                                         |
| $(\lambda y.t)[x/u]$     | $\longrightarrow_{\text{Lamb}}$    | $\lambda y.t[x/u]$                        |                                                                                              |
| $t[x/u][y/v]$            | $\longrightarrow_{\text{Comp}}$    | $t[x/u[y/v]]$                             | $y \in \text{FV}(u)$                                                                         |
| $W_x(t)[x/u]$            | $\longrightarrow_{\text{Weak}_1}$  | $W_{\text{FV}(u)}(t)$                     |                                                                                              |
| $W_y(t)[x/u]$            | $\longrightarrow_{\text{Weak}_2}$  | $W_y(t[x/u])$                             | $x \neq y$                                                                                   |
| $C_x^{y,z}(t)[x/u]$      | $\longrightarrow_{\text{Cont}_1}$  | $C_{\Phi}^{\Delta, \Pi}(t[y/u_1][z/u_2])$ | where<br>$\Phi := \text{FV}(u)$<br>$u_1 = R_{\Delta}^{\Phi}(u)$<br>$u_2 = R_{\Pi}^{\Phi}(u)$ |
| System r                 |                                    |                                           |                                                                                              |
| $W_y(t)\ u$              | $\longrightarrow_{\text{W App}_1}$ | $W_y(t\ u)$                               |                                                                                              |
| $t\ W_y(u)$              | $\longrightarrow_{\text{W App}_2}$ | $W_y(t\ u)$                               |                                                                                              |
| $\lambda x.W_y(t)$       | $\longrightarrow_{\text{W Lamb}}$  | $W_y(\lambda x.t)$                        | $x \neq y$                                                                                   |
| $t[x/W_y(u)]$            | $\longrightarrow_{\text{W Subs}}$  | $W_y(t[x/u])$                             |                                                                                              |
| $C_w^{y,z}(W_y(t))$      | $\longrightarrow_{\text{Merge}}$   | $R_w^z(t)$                                |                                                                                              |
| $C_w^{y,z}(W_x(t))$      | $\longrightarrow_{\text{Cross}}$   | $W_x(C_w^{y,z}(t))$                       | $x \neq y, z$                                                                                |
| $C_w^{y,z}(t\ u)$        | $\longrightarrow_{\text{C App}_1}$ | $C_w^{y,z}(t)\ u$                         | $y, z \in \text{FV}(t)$                                                                      |
| $C_w^{y,z}(t\ u)$        | $\longrightarrow_{\text{C App}_2}$ | $t\ C_w^{y,z}(u)$                         | $y, z \in \text{FV}(u)$                                                                      |
| $C_w^{y,z}(\lambda x.t)$ | $\longrightarrow_{\text{C Lamb}}$  | $\lambda x.C_w^{y,z}(t)$                  |                                                                                              |
| $C_w^{y,z}(t[x/u])$      | $\longrightarrow_{\text{C Subs}}$  | $t[x/C_w^{y,z}(u)]$                       | $y, z \in \text{FV}(u)$                                                                      |

**Figure 7.3:** Reduction rules for  $\lambda\text{lxr}$

unique free occurrence and where each binder binds exactly one occurrence of a variable. Any pure term can be represented as a linear  $\lambda$ l<sub>xr</sub> term using weakenings and contractions. This linearity appears to be a large factor in allowing  $\lambda$ l<sub>xr</sub> to retain PSN whilst having full composition of substitutions (FCS). Substitutions are also never needlessly copied; the **Cont**<sub>1</sub> rule which copies substitutions in  $\lambda$ l<sub>xr</sub> does so conditionally and out of necessity – there is always guaranteed to be exactly two free occurrences of  $y$  and  $z$  below the contraction  $C_x^{y,z}$ . In comparison, the  $\rightarrow_{\text{App}_1}$  rule in  $\lambda$ xgc duplicates substitutions unconditionally and possibly unnecessarily as free occurrences of the bound variable may not exist in either branch of the application. This mix of unconditional copying of substitutions and distributive local substitution rules can break PSN; see Section 7.3.

**Definition** ( $\lambda$ l<sub>xr</sub> reduction).  *$\lambda$ l<sub>xr</sub> reduction, or  $\rightarrow_{\lambda\text{l}_{\text{xr}}}$ , is defined as the union of the rules in Figure 7.3 modulo the smallest congruence generated by the axioms<sup>2</sup> of Figure 7.2.*

The congruence relation allows the use of the meta-notation  $W_Y(M)$  and  $C_{\Theta}^{\Delta, \Pi}$  where  $Y$  and  $\Theta$  are lists of variables and  $\Delta$  and  $\Pi$  are lists of fresh variables isomorphic to  $\Theta$  *i.e.* the order of adjacent weakenings is unimportant and adjacent contractions may be rearranged. Given two finite lists  $\Theta$  and  $\Gamma$  of distinct variables and equal size  $n$ , a renaming operation  $R_{\Gamma}^{\Theta}(M)$  is defined as the result of simultaneously substituting the  $i^{\text{th}}$  variable of  $\Gamma$  for every free occurrence of the  $i^{\text{th}}$  variable of  $\Theta$  in  $M$  where  $i \in 1, \dots, n$ . For example,  $R_{u,v}^{x,y}(\lambda x.xy) = \lambda x.xv$ .

The rewrite relation seems complex but many of the reduction rules of  $\lambda$ l<sub>xr</sub> (especially in System r) deal with pulling weakenings outwards and pushing contractions inwards. Linearity of terms means that substitutions are not replicated during propagation through a term unless a contraction is reached in which case the substitution is duplicated and the copies renamed to maintain linearity **Cont**<sub>1</sub>. Besides these rules, the main familiar ones are substitution introduction **b**, copying **Var**, and explicit garbage collection **Weak**<sub>1</sub>. There is one reduction rule for explicit composition of substitutions **Comp**. This rule only takes care of the case  $y \in \text{FV}(P)$  but the other case  $y \in \text{FV}(t)$  is taken care of by the  $\equiv_S$  congruence. This allows  $\lambda$ l<sub>xr</sub> FCS.

We will use this property of  $\lambda$ l<sub>xr</sub> in the following chapter.

**Lemma 7.1.**  *$\rightarrow_{\text{xr}}$  is strongly normalising [79].*

<sup>2</sup>The congruence axioms were chosen to strengthen the relationship between  $\lambda$ l<sub>xr</sub> and Proof Nets.

|                      |                                   |                    |                                             |                                                                                                                                                                               |
|----------------------|-----------------------------------|--------------------|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $(\lambda x.t)u$     | $\longrightarrow_b$               | $t[x/u]$           |                                             | (subs. generation)                                                                                                                                                            |
| $x[x/t]$             | $\longrightarrow_{\text{Var}}$    | $t$                |                                             |                                                                                                                                                                               |
| $(t\ u)[y/v]$        | $\longrightarrow_{\text{App}_1}$  | $t[x/v]\ u[x/v]$   | $x \in \text{FV}(t), x \in \text{FV}(u)$    | <div style="display: flex; align-items: center;"> <div style="font-size: 3em; margin-right: 5px;">}</div> <div>             local<br/>             substitution </div> </div> |
| $(t\ u)[x/v]$        | $\longrightarrow_{\text{App}_2}$  | $t\ u[x/v]$        | $x \notin \text{FV}(t), x \in \text{FV}(u)$ |                                                                                                                                                                               |
| $(t\ u)[x/v]$        | $\longrightarrow_{\text{App}_3}$  | $t[x/v]\ u$        | $x \in \text{FV}(t), x \notin \text{FV}(u)$ |                                                                                                                                                                               |
| $(\lambda x.t)[y/u]$ | $\longrightarrow_{\text{Lamb}}$   | $\lambda x.t[y/u]$ |                                             |                                                                                                                                                                               |
| $t[x/u][x/v]$        | $\longrightarrow_{\text{Comp}_1}$ | $t[y/v][x/u[y/v]]$ | $y \in \text{FV}(t), y \in \text{FV}(u)$    |                                                                                                                                                                               |
| $t[x/u][y/v]$        | $\longrightarrow_{\text{Comp}_2}$ | $t[x/u[y/v]]$      | $y \notin \text{FV}(t), y \in \text{FV}(u)$ |                                                                                                                                                                               |
| $t[x/u]$             | $\longrightarrow_{\text{gc}}$     | $t$                | $x \notin \text{FV}(t)$                     | (garbage collection)                                                                                                                                                          |

**Figure 7.4:** Reduction rules for  $\lambda\text{es}$ 

### 7.2.3 $\lambda\text{es}$

$\lambda\text{es}$  [78] is a calculus enjoying all the desirable properties listed in Section 2.6 making it possibly unique in this respect to date<sup>3</sup>. It does not have explicit contractions or weakenings like  $\lambda\text{lxr}$  and its congruence and reduction relations are much simpler. As with  $\lambda\text{lxr}$ , terms can be translated into proof-nets such that reduction is weakly simulated, allowing normalisation properties of  $\lambda\text{es}$  to be derived from proof-nets.

$\lambda\text{es}$  uses the set of terms  $\Lambda x$  with usual definitions of free and bound variables. Reduction, like  $\lambda\text{lxr}$ , is modulo an equivalence relation albeit a much simpler one as contractions and weakenings are not present in  $\lambda\text{es}$ .

**Definition** ( $\lambda\text{es}$  equivalence). *Terms of  $\lambda\text{es}$  are considered equivalent up to smallest equivalence relation, denoted  $\mathbf{E}_s$ , containing  $\alpha$ -conversion and the relation*

$$t[x/v][y/u] \equiv_S t[y/u][x/v] \quad \text{if } y \notin \text{FV}(v) \ \& \ x \notin \text{FV}(u).$$

**Definition** ( $\lambda\text{es}$  reduction).  *$\lambda\text{es}$  reduction, or  $\longrightarrow_{\lambda\text{es}}$ , is defined as the union of the rules in Figure 7.4 modulo  $\mathbf{E}_s$  equivalence. The reduction relation  $\longrightarrow_{\text{es}}$  is defined as the union of the local substitution and garbage collection rules modulo  $\mathbf{E}_s$  equivalence.*

<sup>3</sup>Since the time of writing, joint work with Delia Kesner has proven that  $\Lambda_{\text{sub}}$  can be extended with metavariables such that open confluence holds.

|                  |                    |             |                                                           |
|------------------|--------------------|-------------|-----------------------------------------------------------|
| $(\lambda x.t)u$ | $\rightarrow_b$    | $t[x/u]$    | (substitution generation)                                 |
| $C[x][x/u]$      | $\rightarrow_c$    | $C[u][x/u]$ | where this $x$ is free in $C[x]$ (non-local substitution) |
| $t[x/u]$         | $\rightarrow_{gc}$ | $t$         | if $x \notin \text{FV}(t)$ (garbage collection)           |

**Figure 7.5:** Reduction rules for  $\Lambda_{\text{sub}}$ 

### 7.2.4 $\Lambda_{\text{sub}}$

$\Lambda_{\text{sub}}$  also uses the set of terms  $\Lambda x$  where the bound and free variables of terms are as usual<sup>4</sup>.

**Definition** ( $\Lambda_{\text{sub}}$  reduction).  $\Lambda_{\text{sub}}$  reduction, or  $\rightarrow_{\text{bcgc}}$ , is defined as the union of the rules in Figure 7.5.

$\Lambda_{\text{sub}}$  differs from the previous calculi in its propagation calculus which is non-local; the  $\rightarrow_c$  reduction replaces a free occurrence of a variable  $x$  somewhere below a substitution  $[x/u]$  with the subterm  $u$ . We call  $\rightarrow_c$  *wide* or *non-local substitution* as the corresponding parametric reaction rule of  $\lambda\text{BIG}$  is wide; it takes a bigraph with width two as its parameter. Milner describes such rules as *non-local* and  $\rightarrow_c$  as substitution acting ‘at a distance’ [109, 110]. Klop has called similar rules substitutions in term graph rewrite systems *external substitutions* [84].

In a  $\rightarrow_c$  redex, the free occurrence of the variable may be located apart from the substitution definition. This is in contrast to the distributive, *local* rules of most explicit substitution calculi. This property is what makes  $\Lambda_{\text{sub}}$  interesting. While it seems an odd rule to define from a traditional viewpoint, it seems intuitive in the bigraphical setting of  $\lambda\text{BIG}$ .

This non-local substitution allows full composition of substitutions in  $\Lambda_{\text{sub}}$ , a feature which we will discuss particularly in the proof of PSN. For example,

$$(xyx)[y/z][z/w] \rightarrow_c (xzx)[y/z][z/w] \rightarrow_c (xzx)[y/w][z/w] \rightarrow_c (xwx)[y/w][z/w]$$

is a valid reduction sequence in  $\Lambda_{\text{sub}}$ .

We will show later that  $\Lambda_{\text{sub}}$  can simulate  $\beta$ -reduction step-by-step. For now, it may be helpful to accept that if  $(\lambda x.t) u \rightarrow_\beta t\{x/u\}$  where  $t$  has  $n$  free occurrences of  $x$  then  $t \rightarrow_b \rightarrow_c^n \rightarrow_{gc} u$ . In the following lemma, the function  $\mathbb{C}$  (Definition 9.29) ‘reverts’ a term to a pure descendant.

**Lemma 7.2.** In  $\lambda\text{xgc}$ ,  $\lambda\text{es}$ , and  $\Lambda_{\text{sub}}$ ,  $\mathbb{C}(t) \rightarrow_b t$  for all terms  $t$ .

<sup>4</sup>Milner does not explicitly state the set of free variables of terms but they can be teased out of the translation from  $\Lambda_{\text{sub}}$  to  $\lambda\text{BIG}$ .

### 7.2.5 Comparing $\Lambda_{\text{sub}}$ and $\lambda\text{xgc}$

$\Lambda_{\text{sub}}$  is based upon the  $\lambda\sigma$  calculus of Abadi et al. [1]. However, we initially found that Milner's calculus had more in common with  $\lambda\text{xgc}$ .  $\Lambda_{\text{sub}}$  is a named calculus and has a wide, explicit garbage collection rule.  $\lambda\sigma$  lacks this rule and although a named variant of  $\lambda\sigma$  was discussed by Abadi et al., there are some complications [1, 132]. As a result,  $\lambda\sigma$  is presented primarily using de Bruijn notation.

On the other hand,  $\lambda\text{xgc}$  is a named calculus sharing the same set of terms as  $\Lambda_{\text{sub}}$ . The definition of free variables is identical and the only difference between their reduction relations lies in the substitution rules (although the normal forms of  $\rightarrow_x$  and  $\rightarrow_{\text{cgc}}$  coincide). This difference is important but there are enough similarities between the two calculi to prove confluence of  $\Lambda_{\text{sub}}$  using Bloo and Rose's methods; we do so in the following chapter.

The main difference between the two calculi is that  $\Lambda_{\text{sub}}$  also allows full composition of substitutions whereas  $\lambda\text{xgc}$  does not allow any composition<sup>5</sup>. As a result, the set of strongly normalising terms of  $\Lambda_{\text{sub}}$  is a subset of that of  $\lambda\text{xgc}$  as we will see later.

### 7.2.6 Comparing $\Lambda_{\text{sub}}$ and $\lambda\text{lrx}$

The similarities between  $\Lambda_{\text{sub}}$  and  $\lambda\text{xgc}$  helped find a proof of confluence for  $\Lambda_{\text{sub}}$ . However, Bloo and Rose's techniques were harder to apply to a proof of PSN for  $\Lambda_{\text{sub}}$ . The main reason was that it was harder for us to apply induction over the non-local  $\rightarrow_c$  rule than it was over the distributive  $\rightarrow_x$  relation which lends itself easily to inductive methods. We presented an inductive proof of PSN in previous work [122] but we do not find it very convincing in hindsight<sup>6</sup>. The non-local nature of the  $\rightarrow_c$  rule also seemed to make other techniques for proving PSN of  $\lambda\text{xgc}$  like the recursive path ordering technique [18, 19] difficult for  $\Lambda_{\text{sub}}$ .

$\lambda\text{lrx}$  shares an important property with  $\Lambda_{\text{sub}}$  – full composition of substitutions. It also has the PSN property. This gave us the idea of proving PSN of  $\Lambda_{\text{sub}}$  by means of a simulation between the two calculi where a reduction step in  $\Lambda_{\text{sub}}$  corresponded to a non-empty reduction sequence in  $\lambda\text{lrx}$ . This method is only possible using a calculus with both FCS and PSN.

The main difference between the two calculi is the linearity of  $\lambda\text{lrx}$ .

For historical reasons, we note that the two calculi were created independently around the same period; the draft paper in which  $\Lambda_{\text{sub}}$  originated dates to 2004.

<sup>5</sup>Extensions of  $\lambda\text{xgc}$  do – see Sections 7.3.3-7.3.5.

<sup>6</sup>We tried to force Bloo and Rose's techniques to solve our problem and relied on complicated predicates. The result was unsatisfactory for a number of reasons and rather than revisit or revise it (or subject the reader to it) we preferred to find a better solution.



### 7.2.7 Comparing $\Lambda_{\text{sub}}$ and $\lambda\text{es}$

$\lambda\text{es}$  seems the closest match to  $\Lambda_{\text{sub}}$  in the literature. It shares the same set of terms, has FCS, closed confluence, simulation of  $\beta$ -reduction, and PSN.

Delia Kesner suggested proving PSN for  $\Lambda_{\text{sub}}$  using a simulation involving  $\lambda\text{es}$  instead of the more complex  $\lambda\text{lxr}$ . This led to the joint work presented in Chapter 9 where it is shown using intersection type systems that the sets of strongly normalising terms of both calculi coincide.

An simple extension of  $\Lambda_{\text{sub}}$  retains the previously mentioned properties and is confluent on metaterms (open confluence) like  $\lambda\text{es}$  [81]. It seems that  $\Lambda_{\text{sub}}$  and  $\lambda\text{es}$  essentially differ only in whether substitution is local or non-local.

## 7.3 PSN and composition of substitutions

In this section, we discuss how composition of substitutions in explicit substitution calculi may affect the PSN property and so present an intuition of why this property holds for  $\Lambda_{\text{sub}}$ .

Consider the  $\beta$ -reduction path

$$(\lambda x.(\lambda y.xyv)(zw))u \rightarrow_{\beta} (\lambda y.uyv)(zw) \rightarrow_{\beta} u(zw)v.$$

In  $\lambda\text{xgc}$  and  $\Lambda_{\text{sub}}$ , we have a reduction path

$$(\lambda x.(\lambda y.xyv)(zw))u \rightarrow ((\lambda y.xyv)(zw))[x/u] \rightarrow (xyv)[y/zw][x/u].$$

In  $\lambda\text{xgc}$ , the outermost substitution  $[x/u]$  cannot be applied while it is above  $[y/zw]$  *i.e.* first  $[y/zw]$  must be pushed inside the term  $xyv$  and either applied or garbage-collected. In other words, substitutions are *blocked* by substitutions below them. We would like to allow substitutions to interact in some way by either swapping them with a rule like

$$t[x/u][y/v] \longrightarrow t[y/v][x/u[y/v]] \tag{7.1}$$

or by allowing them to compose like

$$t[x/u][y/v] \longrightarrow t[x/u[y/v]] \quad \text{if } y \notin \text{FV}(t)^7 \tag{7.2}$$

which seems efficient if  $x \in \text{FV}(\downarrow_{\text{cgc}}(t))$  and  $y \in \text{FV}(u)$ . Substitution calculi with such rules are said to allow *composition of substitutions*. The rules above are what we call *explicit composition*. We define explicit composition to be composition via a reduction rule whose only purpose is to

<sup>7</sup>The condition is necessary: consider  $(yx)[x/u][y/v] \longrightarrow (yx)[x/u[y/v]]$ .

compose substitutions. Rule (7.1) above is obviously unsafe for PSN as it immediately allows infinite paths of explicit compositions. In this section, we recall – using Bloo’s examples [18] – how the explicit nature of seemingly safe rules like (7.2) may break PSN for certain calculi.

### 7.3.1 Weak/full composition

A reduction system with Rule (7.1) above has FCS. However, as this rule is unsafe for termination and PSN, most calculi with FCS do not use it.

*Weak composition of substitutions* (WCS) is defined as conditional composition – composition may occur but only conditionally. A reduction system where Rule (7.2) above was the only rule for composition would have WCS.

$\lambda xgc$  does not have any rule for composing substitutions. Extensions to  $\lambda xgc$  with FCS and WCS are discussed in the following sections. We also explain how  $\Lambda_{\text{sub}}$  has FCS but how the composition is not explicit.

### 7.3.2 Breaking PSN

Melliès’ counterexample [98] for PSN of  $\lambda\sigma$  demonstrated that this seemingly natural property need not hold.  $\lambda\sigma$  allows substitutions to be explicitly and fully composed. This composition combined with the distributive rules for pushing substitutions inside terms is the essence of Melliès’ counterexample. Although  $\lambda\sigma$  allows parallel substitutions, Bloo notes that this parallelism is not what breaks PSN but rather:

*“the essential property for losing PSN is the possibility of moving one substitution from outside a second substitution to the inside of the latter by means of a composition of substitutions. [18]”*

He shows this by first considering a similar calculus  $\lambda x||c$  with parallel substitutions and full composition and demonstrating that PSN is broken. He then drops the parallel construct to define a calculus  $\lambda xc$  with weak composition. The counterexample for PSN in  $\lambda xc$  is similar to that of  $\lambda x||c$  implying that parallel substitutions is not the essential property for losing PSN.

We will revisit Bloo’s counterexamples for  $\lambda x||c$  and  $\lambda xc$  below, pointing out how the distributive rules and explicit composition are crucial for breaking PSN. We informally discuss Melliès’ counterexample in the same light and then return our attention to  $\Lambda_{\text{sub}}$  which has no distributive rules or explicit composition.

### 7.3.3 $\lambda x||c$

$\lambda x||c$  [20, 18] is an extension of  $\lambda xgc$  with parallel substitutions and FCS. The terms are defined inductively as:

$$t ::= x \mid \lambda x.t \mid t \ t \mid t[x_1, \dots, x_m/t_1, \dots, t_m].$$

The final term structure is an explicit parallel substitution where  $m > 0$  and the variable convention applies *e.g.* the bound variables  $x_1, \dots, x_m$  are assumed to be distinct. The substitution is interpreted as a simultaneous substitution of  $t_i$  for  $x_i$ ,  $1 \leq i \leq m$  in the term  $t$ . The abbreviation  $t[\vec{x}/\vec{u}]$  denotes  $t[x_1, \dots, x_m/u_1, \dots, u_m]$ .

The reduction relation in  $\lambda x||c$  is the union of  $\rightarrow_b$ ,  $\rightarrow_{x||}$ , and  $\rightarrow_{||c}$ . The relation  $\rightarrow_{x||}$  is defined similarly to  $\rightarrow_x$ . The new reduction is  $\rightarrow_{||c}$ , defined as the contextual closure of

$$t[\vec{x}/\vec{u}][\vec{y}/\vec{v}] \rightarrow_{||c} t[\vec{x}, \vec{y}/u_1[\vec{y}/\vec{v}], \dots, u_m[\vec{y}/\vec{v}], \vec{v}].$$

This rule adds FCS to  $\lambda xgc$ .

We now aim to give an intuition as to why the simply typable term  $t$  below, strongly normalising for  $\beta$ -reduction, is not strongly normalising in  $\lambda x||c$ . The full example was given by [18]. In this example, we forget the variable convention to concentrate on the pattern of reduction. The reader may consider all bound variables in the terms to be subscripted with unique numbers.

$$t \equiv \lambda y.(\lambda x.(\lambda x.y)y)((\lambda x.y)y)$$

$$\rightarrow_b \quad \lambda y.((\lambda x.y)y)[x/(\lambda x.y)y] \quad (1)$$

$$\rightarrow_{x||} \quad \lambda y.(\lambda x.y[x/(\lambda x.y)y])(y[x/(\lambda x.y)y]) \quad (2)$$

$$\rightarrow_b \quad \lambda y.y[x/(\lambda x.y)y][x/y[x/(\lambda x.y)y]] \quad (3)$$

$$\rightarrow_{||c} \quad \lambda y.y[x, x'/\underline{((\lambda x.y)y)[x/y[x/(\lambda x.y)y]]}, y[x/(\lambda x.y)y]] \quad (4)$$

The reduction sequence can be described as follows. The substitution in (1) is pushed inside the term by the distributive rule  $\rightarrow_{x||}$ . The pushing of the substitution inside the application/ $\rightarrow_b$ -redex  $(\lambda x.u)u$  duplicates the substitution (2). The  $\rightarrow_b$  redex then fires, leaving the both copies of the substitution side by side (3). Note that the outer copy contains the original substitution. The copies are then composed (4).

The underlined subterm in (4) has an infinite reduction path which follows a similar pattern: the substitution is pushed inside the  $\rightarrow_b$  redex, copying itself; the redex fires; the substitutions are repeatedly composed until one lies just outside a  $\rightarrow_b$  redex; the process repeats.

It is important to note that the infinite path above is made possible by the interplay between the distributive rules (which duplicate substitutions), the  $\rightarrow_b$  rule which places a substitution

beside its descendant, and the composition rule which places a substitution inside its descendant. This last feature seems essential for losing PSN in calculi with explicit composition [18, p.60].

We note that these infinite paths do not involve any substitutions being performed or garbage-collected. It is the distributive nature of the  $\rightarrow_{x||}$  and  $\rightarrow_x$  rules which require substitutions to be needlessly copied which can be dangerous. By ‘needlessly copied’ we mean that more copies of a substitution  $[x/u]$  may be created than free occurrences of  $x$  exist or that copies of a substitution are made whether or not free occurrences of  $x$  exist below all of the copies. Wide substitution therefore seems an important concept as its use avoids this creation of needless copying. Another solution to this problem is to always have exactly one free occurrence of a variable below a substitution so that the substitution is never needlessly copied. This is the approach taken in  $\lambda\text{lxr}$  which uses this linearity to keep PSN and FCS.

#### 7.3.4 $\lambda\text{xc}$

We have seen above that composition of substitutions can lead to infinite paths involving the distributive rules and substitution creation. However, the parallelism of substitutions in  $\lambda x||c$  does not seem to be the important factor for losing PSN. Bloo and Rose [20, 18] made this intuition precise by introducing  $\lambda\text{xc}$ , which can be viewed as  $\lambda x||c$  without parallelism.

$\lambda\text{xc}$  shares the same set of terms as  $\lambda\text{xgc}$ . The reduction relation of  $\lambda\text{xc}$  is the union of  $\rightarrow_{\text{bxgc}}$  and  $\rightarrow_{c_1}$ , where  $\rightarrow_{c_1}$  is defined as the contextual closure of

$$t[x/u][y/v] \rightarrow_{c_1} t[x/u[y/v]] \quad \text{if } y \notin \text{FV}(t).$$

This rule adds WCS to  $\lambda\text{xgc}$  and is in fact the rule (7.2) introduced at the beginning of this section. It seems an efficient rule for the case where  $x \in \text{FV}(\downarrow_{\text{xgc}}(t))$ ,  $y \in \text{FV}(u)$ .

Bloo [18] shows that PSN is broken in  $\lambda\text{xc}$  in a similar fashion to  $\lambda x||c$ .

$$\begin{aligned} t &\equiv \lambda y.(\lambda x.(\lambda x.y)y)((\lambda x.y)y) \\ \rightarrow_b \quad &\lambda y.((\lambda x.y)y)[x/(\lambda x.y)y] \quad (1) \\ \rightarrow_x \quad &\lambda y.(\lambda x.y[x/(\lambda x.y)y])(y[x/(\lambda x.y)y]) \quad (2) \\ \rightarrow_b \quad &\lambda y.y[x/(\lambda x.y)y][x/y[x/(\lambda x.y)y]] \quad (3) \\ \rightarrow_{c_1} \quad &\lambda y.y[x/\underline{((\lambda x.y)y)}][x/y[x/(\lambda x.y)y]] \quad (4) \end{aligned}$$

The infinite reduction begins in the underlined subterm in a similar manner. Again, the interplay between  $\rightarrow_{\text{App}_1}$ ,  $\rightarrow_{\text{Lamb}}$ ,  $\rightarrow_b$ , and the composition rule breaks PSN.

This counterexample leads one to believe that in order to have PSN, an explicit substitution calculus should not create subterms inside substitutions which cannot be created outside substitutions. Bloo notes this in his dissertation and this intuition was the essence of our inductive proof of PSN for  $\Lambda_{\text{sub}}$  [122]<sup>8</sup>. Armed with this intuition, Bloo and Geuvers [19] further constrained the composition of  $\lambda\text{xc}$  and were able to show that the new calculus  $\lambda\text{xc}^-$  satisfied PSN.

### 7.3.5 $\lambda\text{xc}^-$

$\lambda\text{xc}^-$  shares the same set of terms as  $\lambda\text{xgc}$ . The reduction relation of  $\lambda\text{xc}^-$  is the union of  $\rightarrow_{\text{bxgc}}$  and  $\rightarrow_{c^-}$ , where  $\rightarrow_{c^-}$  is defined as the contextual closure of

$$t[x/u][y/v] \rightarrow_{c^-} t[x/u][y/v] \quad \text{if } x \in \text{FV}(\downarrow_x(t)), y \notin \text{FV}(t).$$

This calculus is confluent, preserves strong normalisation, and has WCS [19].

### 7.3.6 $\lambda\sigma$

We will not detail Mellies' counterexample for  $\lambda\sigma$  as we would have to introduce too much extra notation. However, the counterexample may be described as follows, where all rules are in  $\lambda\sigma$ .

Applications of the *Beta* rule (similar to the  $\rightarrow_b$  rule) create explicit substitutions. The *App* and *Lamb* rules distribute these substitutions inside the term as in the examples above. A *Beta* rule then creates a new substitution above the original one. The rules *Clos*, *Map*, and *Ass* then compose these two substitutions. Reduction sequences of this form continue indefinitely.

Again, the same interplay of similar rules yields the counterexample.

### 7.3.7 $\Lambda_{\text{sub}}$

As  $\Lambda_{\text{sub}}$  satisfies PSN, it seems natural to investigate how it allows composition of substitutions. Although there is no explicit composition rule, it has FCS – a substitution can always be performed whenever a free occurrence of a variable lies beneath the substitution definition regardless of what lies between. Two adjacent substitutions may also be *implicitly* composed.

By *implicit*, we mean the following. In  $\lambda\text{xgc}$ , we could read  $t[x/u][y/v]$  as ‘replace  $x$  with  $u$  in  $t$  then  $y$  with  $v$  in the result’ as the inner substitution must be applied or discarded before the outer substitution can be performed. In  $\Lambda_{\text{sub}}$ , we can read the same term as ‘replace  $x$  with  $u$  in  $t$  or  $y$

<sup>8</sup>As we explain in the next chapter, this inductive proof is complicated and possibly unconvincing which is why we do not present it here.

with  $v$  in  $t$  or  $u$ .’ Composition is allowed but not via an explicit reduction rule – it is allowed as  $\rightarrow_c$  is a wide or non-local rule.

In  $\Lambda_{\text{sub}}$ , we also have reduction sequences like:

$$\begin{aligned} & t[x/u][y/v] \\ \rightarrow_c & t\{y/v\}[x/u\{y/v\}][y/v] \\ \rightarrow_{\text{gc}} & t\{y/v\}[x/u\{y/v\}] \end{aligned}$$

where  $t\{y/v\}$  means ‘the  $\Lambda x$  term  $t$  with all free occurrences of  $y$  replaced by  $v$ ’. This sequence demonstrates how Rule (7.1) can be mimicked via wide substitution whilst avoiding the obvious infinite reductions. This is how Kesner [78] defines FCS and is the definition we took earlier.

Wide substitution is a very useful feature as it avoids the copying of a substitution definition using a distributive rule like  $\rightarrow_{\text{App1}}$ . This removes “*the possibility of bringing a substitution into a descendant of itself*” [18] which leads to the counterexamples above. In fact, in the examples above PSN was broken without substitutions ever being performed. This is impossible in  $\Lambda_{\text{sub}}$  as  $\rightarrow_{\text{bgc}}$  is strongly normalising. This substitution ‘at a distance’ allows  $\Lambda_{\text{sub}}$  to have FCS and PSN.

## 7.4 Summary and related work

We have introduced a number of explicit substitution calculi which we have identified as being useful for proving confluence and PSN for  $\Lambda_{\text{sub}}$  in the forthcoming chapters. The problems that composition of substitutions can introduce with respect to preserving strong normalisation of  $\beta$ -reduction were discussed and we found that although  $\Lambda_{\text{sub}}$  allows full composition, it is counter-intuitive – based on existing intuitions as to what breaks PSN – to assume that it lacks PSN. This prepares us for the proofs of PSN in the following chapters.

Summaries of explicit substitution calculi with respect to the properties of Definition 2.54 have been published throughout the literature so we will not reproduce them here. Kesner [78] summarises the current state of the art and points out the shortcomings of calculi without full composition; Bloo’s summary [18] highlights the progress made in the past decade. To date, only one calculus,  $\lambda\text{es}$ , satisfies all of those properties. All properties bar open confluence have been proven for  $\lambda\text{lxr}$  [80] and  $\Lambda_{\text{sub}}$  (here). Open confluence has not been disproven for  $\lambda\text{lxr}$  and it has been proven for a simple extension of  $\Lambda_{\text{sub}}$  [81]. That work also relates  $\Lambda_{\text{sub}}$  to calculi with partial substitutions and calculi with definitions.

# Chapter 8

## Inductive Proofs

*Don't believe me if I tell you*

*Not a word of this is true*

*Don't Believe A Word – Thin Lizzy*

The similarities between  $\Lambda_{\text{sub}}$  and  $\lambda\text{xgc}$  led us to ask whether the proofs of confluence and preservation of strong normalisation (PSN) for  $\lambda\text{xgc}$  could be applied to  $\Lambda_{\text{sub}}$ .

The question of confluence is investigated in this chapter. In Section 8.1, we prove that  $\Lambda_{\text{sub}}$  can simulate  $\beta$ -reduction step-by-step and then show that the rewrite relation of  $\Lambda_{\text{sub}}$  is confluent by adapting Bloo and Rose's inductive proof which uses the interpretation method.

Appendix B.2 discusses the problems with naively reusing an existing proof of PSN for  $\lambda\text{xgc}$  in  $\Lambda_{\text{sub}}$ . In the appendix, we prove PSN for some subcalculi of  $\Lambda_{\text{sub}}$  using the notion of garbage-free reduction in order to demonstrate how complicated this method becomes when considering the whole calculus.

### 8.1 Proof of confluence for $\Lambda_{\text{sub}}$

In this section we prove that the reduction relation in  $\Lambda_{\text{sub}}$  (and its bigraphical model  $\Lambda\text{BIG}$  – see Chapter 11) is confluent. The proof is based on Bloo and Rose's work on  $\lambda\text{xgc}$  [20, 132, 18]. This yields an indirect proof of confluence for  $\Lambda\text{BIG}$ . The proofs are based on the correspondences between the normal forms of  $\rightarrow_{\text{xgc}}$  and  $\rightarrow_{\text{cgc}}$ .

The main difference between our proofs and the originals is that in working with the wide substitution of  $\Lambda_{\text{sub}}$ , we cannot use the inductive property that a distributive rule like  $\rightarrow_x$  enjoys. In these cases, we employ contexts in place of the original inductive reasoning.

Some details of the proofs below are abbreviated or omitted to save space. The full proofs have been published previously by the author [122]. For a detailed discussion behind the proof strategy, we refer the reader to Rose's tutorial [132]

**Propositions 8.1.** 1.  $\rightarrow_b$  SN, 2.  $\rightarrow_b \diamond$ , 3.  $\rightarrow_{\text{cgc}}$  SN, 4.  $\rightarrow_{\text{cgc}}$  CR, 5.  $\rightarrow_{\text{cgc}}$  UN

*Proof.*

1. Each  $\Lambda x$  term has finitely many  $\rightarrow_b$ -redexes. Each  $\rightarrow_b$ -reduction decreases the number of  $\rightarrow_b$ -redexes. Hence,  $\rightarrow_b$  SN.
2. We take Rose's proof [132, Proposition 1.1.10.2]<sup>1</sup>.
3. See Appendix B, Proposition B.1.
4.  $\rightarrow_{\text{cgc}}$  LC can be proven by inspecting the cases of the proof of  $\rightarrow_{\text{bcgc}}$  LC given in [107, Propositions 5.8, 5.5].  $\rightarrow_{\text{cgc}}$  SN and  $\rightarrow_{\text{cgc}}$  LC imply  $\rightarrow_{\text{cgc}}$  CR by Newman's lemma [117].
5.  $\rightarrow_{\text{cgc}}$  CR implies  $\rightarrow_{\text{cgc}}$  UN [142, Theorem 1.2.2(i), p. 17].

□

Note that pure terms have no  $\rightarrow_{\text{cgc}}$  redexes and garbage-free terms have no  $\rightarrow_{\text{gc}}$  redexes. The following lemmas concerning unique normal forms will be useful. They are required in some proofs where we cannot avail of the inductive methods that were originally employed.

**Lemmas 8.2** (normal forms). *For all  $\Lambda_{\text{sub}}$  terms  $t$  and  $u$ ,*

1.  $\downarrow_{\text{cgc}}(t \ u) \equiv \downarrow_{\text{cgc}}(t) \downarrow_{\text{cgc}}(u)$
2.  $\downarrow_{\text{cgc}}(\lambda x.t) \equiv \lambda x. \downarrow_{\text{cgc}}(t)$
3.  $\downarrow_{\text{cgc}}((t \ u)[x_1/v_1] \cdots [x_n/v_n]) \equiv \downarrow_{\text{cgc}}(t[x_1/v_1] \cdots [x_n/v_n]) \downarrow_{\text{cgc}}(u[x_1/v_1] \cdots [x_n/v_n])$
4.  $\downarrow_{\text{cgc}}((\lambda x.t)[x_1/v_1] \cdots [x_n/v_n]) \equiv \downarrow_{\text{cgc}}(\lambda x.t[x_1/v_1] \cdots [x_n/v_n]) \equiv \lambda x. \downarrow_{\text{cgc}}(t[x_1/v_1] \cdots [x_n/v_n])$
5. 1, 2, and 4 hold with  $\downarrow_{\text{cgc}}$  replaced by  $\downarrow_{\text{gc}}$

<sup>1</sup>A bigraphical proof would consist of a case split over the possible overlappings. There are two non-trivial possibilities. Either both redexes are independent or one lies entirely within the other. The first case yields the diamond property [107]. It should be trivial to prove the same for the second case using Milner's theorems in that text.



*Proof.* For (1),  $t$  and  $u$  can not interact via  $\rightarrow_{\text{cgc}}$  reductions. (1) then follows as  $\rightarrow_{\text{cgc}}$  UN. (4) follows by the variable convention ( $x \neq x_i, 1 \leq i \leq n$ ) and then an application of (2).  $\square$

**Lemma 8.3** (representation). *For all terms  $t, u$  and variable  $x$ ,*

$$\downarrow_{\text{cgc}}(t[x/u]) \equiv \downarrow_{\text{cgc}}(t)\{x/\downarrow_{\text{cgc}}(u)\}$$

*Proof.* Proof by induction over the number of symbols in  $t, u_1, \dots, u_n$  with the hypothesis

$$\downarrow_{\text{cgc}}(t[x_1/u_1] \cdots [x_n/u_n]) \equiv \downarrow_{\text{cgc}}(t)\{x_1/\downarrow_{\text{cgc}}(u_1)\} \cdots \{x_n/\downarrow_{\text{cgc}}(u_n)\}$$

The proof is broken over the structure of  $t$  and uses Lemma 8.2 and the induction hypothesis [122]. We only present one case here.

**Case**  $t \equiv t_1[y/v]^2$

$$\begin{aligned} & \downarrow_{\text{cgc}}(t[x_1/u_1] \cdots [x_n/u_n]) \\ \equiv & \downarrow_{\text{cgc}}(t_1[y/v][x_1/u_1] \cdots [x_n/u_n]) \\ & (t_1, v, u_1 \dots u_n \text{ has less symbols than } t_1[y/v], u_1 \dots u_n) \\ \stackrel{\text{i.h.}}{\equiv} & \downarrow_{\text{cgc}}(t_1)\{y/\downarrow_{\text{cgc}}(v)\}\{x_1/\downarrow_{\text{cgc}}(u_1)\} \cdots \{x_n/\downarrow_{\text{cgc}}(u_n)\} \\ & (\downarrow_{\text{cgc}}(t_1)\{y/\downarrow_{\text{cgc}}(v)\} \text{ has less symbols than } t_1, v, u_1 \dots u_n) \\ \stackrel{\text{i.h.}}{\equiv} & \downarrow_{\text{cgc}}(t_1[y/v])\{x_1/\downarrow_{\text{cgc}}(u_1)\} \cdots \{x_n/\downarrow_{\text{cgc}}(u_n)\} \end{aligned}$$

$\square$

**Corollary 8.4** (substitution lemma [132]). *In  $\Lambda_{\text{sub}}$ ,  $t[x/u][y/v] \longleftrightarrow_{\text{cgc}} t[y/v][x/u[y/v]]$ .*

*Proof.* Follows from the previous lemma, the  $\lambda$ -calculus substitution lemma, and as  $\downarrow_{\text{cgc}}(t)$  is a pure term for any  $t$ . See Corollary B.2 in Appendix B, page 28.  $\square$

The following is slightly different to the corresponding proof for  $\lambda\text{xgc}$ . In  $\Lambda_{\text{sub}}$ , the only type of reduction which reduces the set of free variables of a term is  $\rightarrow_{\text{gc}}$ . In  $\lambda\text{xgc}$ ,  $\rightarrow_{\text{gc}}$  is not the only reduction with this property –  $\rightarrow_{\text{Vargc}}$  may also reduce the set of free variables.

---

<sup>2</sup>Thanks to Thomas Hildebrandt for explaining this case to me.

**Propositions 8.5.** For any  $t$ ,

1. If  $t \rightarrow_{\text{gc}} u$  then  $\text{FV}(t) \supseteq \text{FV}(u)$  and  $\downarrow_{\text{cgc}}(t) \equiv \downarrow_{\text{cgc}}(u)$ .
2. If  $t \rightarrow_{\text{c}} u$  then  $\text{FV}(t) = \text{FV}(u)$  and  $\downarrow_{\text{cgc}}(t) \equiv \downarrow_{\text{cgc}}(u)$ .
3. If  $t \rightarrow_{\text{b}} u$  then  $\text{FV}(t) = \text{FV}(u)$
4. If  $t$  is garbage-free then  $\text{fv}(t) = \text{fv}(\downarrow_{\text{cgc}}(t))$ .

*Proof.* See Appendix B, Propositions B.3. □

**Proposition 8.6.** For pure  $t$ , 1.  $t \xrightarrow{\text{b}} \blacksquare$   $\xrightarrow{\text{cgc}}$   $\Downarrow$   $\square$  and 2.  $t \xrightarrow{\beta} \blacksquare$   $\xrightarrow{\text{cgc}}$   $\Downarrow$   $\square$

*Proof.* There exists a pure context  $C$  such that  $t \equiv C[(\lambda x.u)v]$  with  $u$  and  $v$  pure i.e.  $u \equiv \downarrow_{\text{cgc}}(u)$ ,  $v \equiv \downarrow_{\text{cgc}}(v)$ .

1. We have  $C[(\lambda x.u)v] \rightarrow_{\beta} C[u\{x/v\}]$  and the sequence

$$C[(\lambda x.u)v] \rightarrow_{\text{b}} C[u[x/v]] \xrightarrow{\text{cgc}} C[\downarrow_{\text{cgc}}(u[x/v])] \stackrel{8.3}{\equiv} C[\downarrow_{\text{cgc}}(u)\{x/\downarrow_{\text{cgc}}(v)\}] \equiv C[u\{x/v\}].$$

2. Follows similarly. □

**Lemmas 8.7** (Projection).

1. For all  $\Lambda_{\text{sub}}$ -terms  $t$ ,
 
$$\begin{array}{ccc} t & \xrightarrow{\text{b}} & t' \\ \downarrow_{\text{cgc}} & & \downarrow_{\text{cgc}} \\ \downarrow_{\text{cgc}}(t) & \xrightarrow[\beta]{\dots\dots\dots} & \downarrow_{\text{cgc}}(t') \end{array}$$
2. For garbage-free  $t$ ,
 
$$\begin{array}{ccc} t & \xrightarrow{\text{b}} & t' \\ \downarrow_{\text{cgc}} & & \downarrow_{\text{cgc}} \\ \downarrow_{\text{cgc}}(t) & \xrightarrow[\beta]{+} & \downarrow_{\text{cgc}}(t') \end{array}$$

*Proof.* We prove 2 by inducting over the structure of  $t$ . We present the non-trivial cases. The rest follow by the induction hypothesis and Lemma 8.2.

**Case**  $t \equiv (\lambda x.u)v, t' \equiv u[x/v]$  :

$$\downarrow_{\text{cgc}}(t) \stackrel{8.2}{\equiv} (\lambda x. \downarrow_{\text{cgc}}(u)) \downarrow_{\text{cgc}}(v) \rightarrow_{\beta} \downarrow_{\text{cgc}}(u)\{x/\downarrow_{\text{cgc}}(v)\} \stackrel{8.3}{\equiv} \downarrow_{\text{cgc}}(u[x/v]) \equiv \downarrow_{\text{cgc}}(t')$$

**Case**  $t \equiv u[x/v], t' \equiv u[x/v']$

By the inductive hypothesis, we know that  $\downarrow_{\text{cgc}}(v) \rightarrow_{\beta}^+ \downarrow_{\text{cgc}}(v')$  and so

$$\downarrow_{\text{cgc}}(u)\{x/\downarrow_{\text{cgc}}(v)\} \rightarrow_{\beta}^+ \downarrow_{\text{cgc}}(u)\{x/\downarrow_{\text{cgc}}(v')\}.$$

since by the fact that  $t$  is garbage free we have that  $x \in \text{fv}(u)$  and by Proposition 8.5.4, it follows that  $x \in \text{fv}(\downarrow_{\text{cgc}}(u))$ .

Now, by application of Lemma 8.3 twice we have

$$\downarrow_{\text{cgc}}(t) \stackrel{8.3}{\equiv} \downarrow_{\text{cgc}}(u)\{x/\downarrow_{\text{cgc}}(v)\} \rightarrow_{\beta}^+ \downarrow_{\text{cgc}}(u)\{x/\downarrow_{\text{cgc}}(v')\} \stackrel{8.3}{\equiv} \downarrow_{\text{cgc}}(t').$$

The proof of 1 is similar except that we do not know in the last case that  $x \in \text{fv}(\downarrow_{\text{cgc}}(u))$  and so we have to use  $\rightarrow_{\beta}$ , allowing the reflexive closure to provide identity ( $\equiv$ ) *i.e.* we do a case split taking either  $x \in \text{FV}(u)$  (proof as above) or  $x \notin \text{FV}(u)$  (in which case  $\downarrow_{\text{cgc}}(t) \equiv \downarrow_{\text{cgc}}(t')$ ).

**Case**  $t \equiv u[x/v], t' \equiv u'[x/v]$ : Similar to the last case. □

Proposition 8.6, Lemma 8.7, and  $\rightarrow_{\text{cgc}}$  UN prove that  $\Lambda_{\text{sub}}$  is a conservative extension of the  $\lambda\beta$ -calculus, with  $\rightarrow_{\text{cgc}}$  as translation.

**Theorem 8.8.** *For pure terms  $t, u$ :  $t \rightarrow_{\text{bcgc}} u \iff t \rightarrow_{\beta} u$*

*Proof.*

**Case  $\Leftarrow$ :** Assume  $t \rightarrow_{\beta} u$ ; we then prove by induction on the length  $n$  of the  $\beta$ -reduction, using Proposition 8.6.2 in each step:

**Case:**  $n = 0$  Trivial

**Inductive hypothesis:**  $n = k$   $t \rightarrow_{\beta}^k u \implies t \rightarrow_{\text{bcgc}} u$

**Case:**  $n = k + 1$

$$t \rightarrow_{\beta}^{k+1} u \implies t \rightarrow_{\beta} t' \rightarrow_{\beta}^k u$$

By Proposition 8.6.2 and i.h., we have  $t \rightarrow_{\beta} t' \rightarrow_{\text{bcgc}} u$ .

**Case  $\Rightarrow$ :** We induct over the length of the  $\rightarrow_{\text{bcgc}}$ -reduction to prove

$$\begin{array}{ccccccc} t & \xrightarrow{\text{bcgc}} & t_1 & \xrightarrow{\text{bcgc}} & \cdots & \xrightarrow{\text{bcgc}} & t_{n-1} & \xrightarrow{\text{bcgc}} & u \\ \parallel & & \downarrow \text{cgc} & & & & \downarrow \text{cgc} & & \parallel \\ t & \xrightarrow{\beta} & \downarrow_{\text{cgc}}(t_1) & \xrightarrow{\beta} & \cdots & \xrightarrow{\beta} & \downarrow_{\text{cgc}}(t_{n-1}) & \xrightarrow{\beta} & u \end{array}$$

Each step in the top of each square above is one of  $\rightarrow_{\text{gc}}$ ,  $\rightarrow_{\text{c}}$ , or  $\rightarrow_{\text{b}}$  and so each square gives rise to one of the following diagrams:

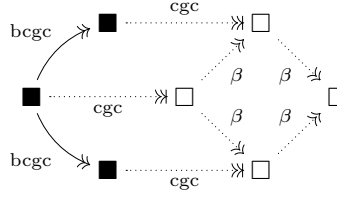
$$\begin{array}{ccc} t_i & \xrightarrow{\text{cgc}} & t_{i+1} \\ \downarrow \text{cgc} & & \downarrow \text{cgc} \\ \downarrow_{\text{cgc}}(t_i) & \equiv & \downarrow_{\text{cgc}}(t_{i+1}) \end{array} \quad \begin{array}{ccc} t_i & \xrightarrow{\text{b}} & t_{i+1} \\ \downarrow \text{cgc} & & \downarrow \text{cgc} \\ \downarrow_{\text{cgc}}(t_i) & \xrightarrow[\beta]{} & \downarrow_{\text{cgc}}(t_{i+1}) \end{array}$$

which follow respectively by  $\rightarrow_{\text{cgc}}$  UN and Lemma 8.7.1.

□

**Corollary 8.9.**  $\rightarrow_{\text{bcgc}}$  CR

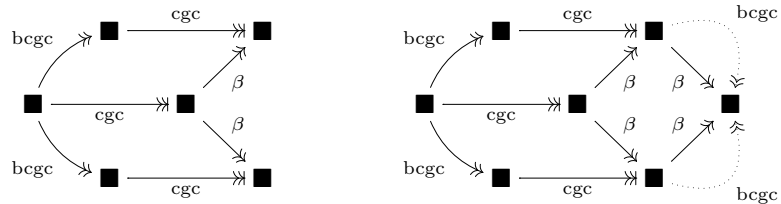
*Proof.* We prove the following diagrammatic proposition which states that  $\rightarrow_{\text{bcgc}}$  is strongly confluent. Strong confluence implies CR.



We have shown that

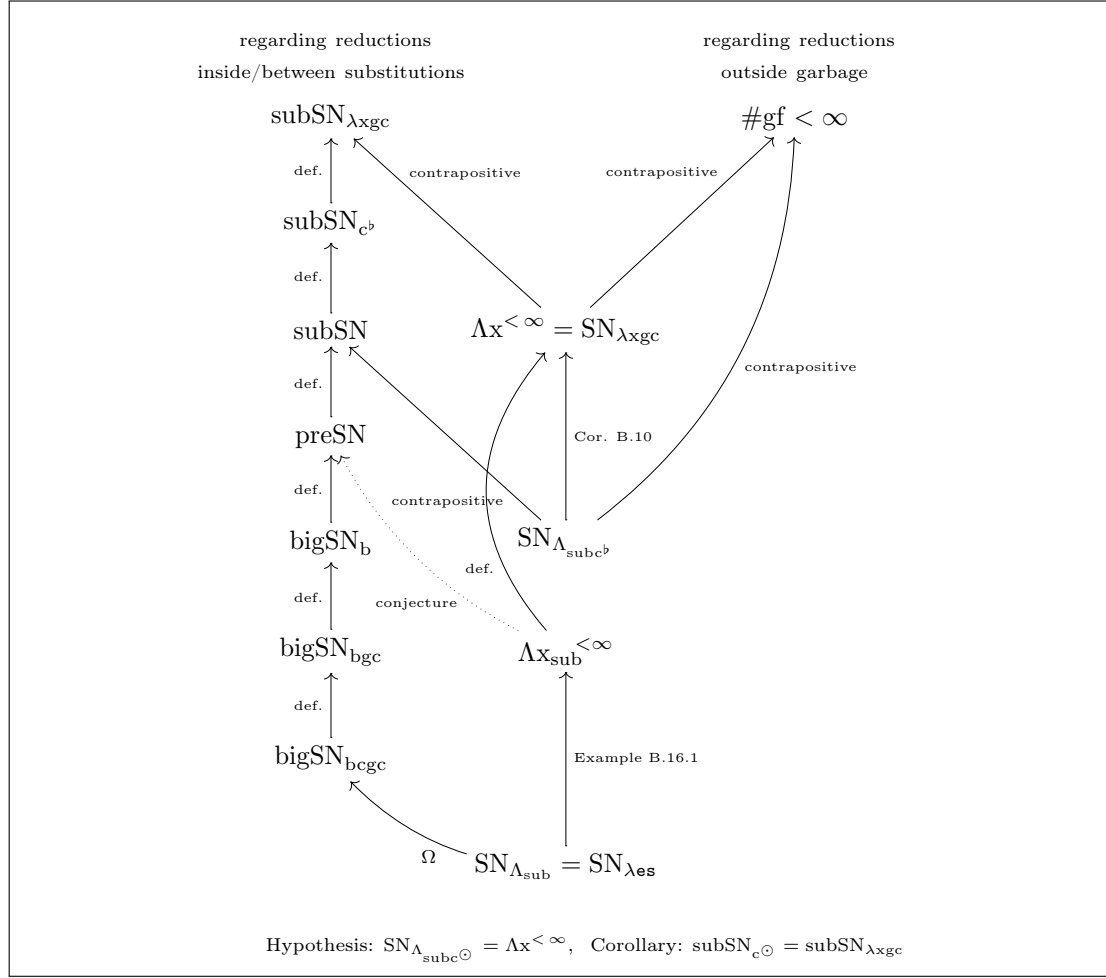
1.  $\rightarrow_{\beta} \subseteq \rightarrow_{\text{bcgc}}$  (by Proposition 8.6.2),
2.  $\forall t \in \Lambda_{\text{sub}} : t \rightarrow_{\text{bcgc}} \downarrow_{\text{cgc}}(t) \in \Lambda$ ,
3.  $\forall t, u \in \Lambda_{\text{sub}} : t \rightarrow_{\text{bcgc}} u \Rightarrow \downarrow_{\text{cgc}}(t) \rightarrow_{\beta} \downarrow_{\text{cgc}}(u)$ . This follows from the fact that a  $t \rightarrow_{\text{bcgc}} u$  reduction is either a  $t \rightarrow_{\text{b}} u$  reduction or a  $t \rightarrow_{\text{cgc}} u$  reduction. The former case is shown by Lemma 8.7.1 and the latter case from  $\rightarrow_{\text{cgc}}$  UN.

The proof follows from the generalised interpretation method [76]: from 2 we can fill in the horizontal arrows on the left diagram below and by 3 we can fill in the diagonal  $\beta$  arrows. By the confluence of  $\rightarrow_{\beta}$  we have the diagram on the right where, by 1, the dotted arrows can be filled in.



Hence,  $\rightarrow_{\text{bcgc}}$  CR.

□

Figure 8.1: Relationship between properties of  $\Lambda x$  terms

## 8.2 Summary and related work

The chapter gave proofs of confluence and step-by-step simulation of  $\beta$ -reduction for  $\Lambda_{sub}$ . As a corollary, we have proven confluence of Milner's  $\lambda BIG$  Brs (see Chapter 11).

By noticing the similarities between  $\Lambda_{sub}$  and  $\lambda xgc$  we were able to reuse Bloo and Rose's proof strategy to prove confluence. Proofs of PSN for subcalculi of  $\Lambda_{sub}$  are similarly presented in Appendix B.2. In the appendix, the difficulties with using Bloo and Rose's approach to prove PSN for  $\Lambda_{sub}$  are highlighted, although this is not surprising as  $\Lambda_{sub}$  has full composition of substitutions.

The relationships between the different subsets of  $\Lambda x$  defined in the appendix and the strongly normalising terms of  $\lambda xgc$ ,  $\lambda es$ , and  $\Lambda_{sub}$  are presented in Figure 8.1 where solid arrows denote subset inclusion. The proofs of the implications are denoted in the figure: all are either proof

by definition, contrapositive, or example. Each subset excludes some  $\Lambda x$  terms which are not strongly normalising for  $\overrightarrow{\text{ACD}}$  or  $\beta$ -reduction. From the diagram, we can see that the addition of levels of inter-substitution reduction (composition of substitutions) decreases the set of strongly normalising terms. We claim that if we disallow any inter-substitution reduction (*i.e.*  $\Lambda_{\text{subc}\odot}$ ) then the resulting calculus is extremely similar to  $\lambda xgc$ . This may be investigated in future work but we think that the proofs should follow from Appendix B.2.2. In previous work [122], we showed that the intersection of  $\text{bigSN}_{\text{bcgc}}$  and  $\#gf$  identify the subset of strongly normalising terms of  $\Lambda_{\text{sub}}$ . However, this could not be considered a simple characterisation. We present such a characterisation in the next chapter using intersection types and show that it is equal to  $\text{SN}_{\lambda\text{es}}$ .

Milner has already given a proof of weak confluence for  $\Lambda_{\text{sub}}$  [107]. It follows from his bigraphical proof in  $\Lambda\text{BIG}$ . While we do not address his challenge of tackling confluence in the bigraphical setting, we prove confluence for  $\Lambda_{\text{sub}}$  yielding an indirect proof for  $\Lambda\text{BIG}$ . Direct, bigraphical proofs of strong confluence for  $\Lambda\text{BIG}$  remain unpublished to date.

Since this work, we have collaborated with Delia Kesner on both a constructive proof of  $\rightarrow_{\text{cgc}}$  SN and an elegant proof of open confluence using Tait and Martin-Löf's technique [81]. These results are largely due to Delia Kesner so we omit the details here.

We discussed how composition of substitutions can break PSN. However, David and Guillaume state that this composition is necessary for open confluence [46]. They point out that  $\lambda_d$  [77] and  $\lambda\sigma_n$  [130] have weak composition of substitutions which is not enough to allow open confluence. It is thus far rare for an explicit substitution calculus to have open confluence, PSN, and FCS; to our knowledge only  $\lambda\text{es}$  and  $\Lambda_{\text{sub}}$  have these properties and the question of open confluence remains open for  $\lambda\text{lxr}$ .

# Chapter 9

## Strong Normalisation

*What some take for magic at first glance*

*Is just sleight of hand depending on what you believe*

*Something gets lost when you translate*

*Invisible Ink – Aimee Mann*

In Chapter 7, we talked about the struggle between composition of substitutions and preservation of strong normalisation. These properties have only recently been reconciled (in  $\lambda_{\text{lr}}$  and  $\lambda_{\text{es}}$ ). Therefore, existing techniques of proving PSN do not seem well-adapted to proving PSN for  $\Lambda_{\text{sub}}$ .

In this chapter, we give proofs of PSN for  $\Lambda_{\text{sub}}$  by simulating reduction in other calculi with PSN. For such a simulation to work, the second calculus requires full composition of substitutions – this leaves two possibilities,  $\lambda_{\text{lr}}$  and  $\lambda_{\text{es}}$ .

Our approach has a novel aspect. The substitution rule in  $\Lambda_{\text{sub}}$  is non-local and explicit substitutions are fixed firmly in place. However in the target calculi, substitution (the **(Var)** rule) is local and explicit substitutions are mobile and propagate through terms. This presents problems for a simulation. Our solution is simply to duplicate (future) bodies of substitution so that one copy stays in place, allowing a syntactic match with the  $\Lambda_{\text{sub}}$  substitution, while the second copy propagates through the term, performing the actual substitutions of terms for variables.

We present an overview of the general idea in Section 9.1. In Section 9.2 we introduce a slightly modified version of  $\lambda_{\text{lr}}$ . This calculus retains PSN (see Appendix B). We then present a proof of

PSN by simulating reduction in this calculus. In Section 9.3 (joint work with Delia Kesner), we proof that  $\Lambda_{\text{sub}}$  reduction can be simulated by reduction in  $\lambda\text{es}$ . This time, we do not modify the target calculus but instead alter the translation so this becomes unnecessary<sup>1</sup>.

The final technical section of this chapter (again joint work) provides a characterisation of  $\text{SN}_{\Lambda_{\text{sub}}}$  using an intersection type system and proves PSN via simulation in  $\lambda\text{es}$ . We explain how the characterisation supports our claim that  $\Lambda_{\text{sub}}$  is dynamically close to the  $\lambda$ -calculus.

## 9.1 Simulating non-local substitution with local substitution

In this section we explain the problems with naively trying to simulate a reduction step in  $\Lambda_{\text{sub}}$  with a non-empty reduction sequence in an ES calculus which propagates substitutions through terms before performing substitution. We then explain our solution, two ways to go about realising it, and the advantages and disadvantages of the two realisations.

We assume a second ES calculus  $\lambda_z$  with the set of terms  $\Lambda x$  and the reduction relation  $\rightarrow_{\lambda_z}$  consisting of the rules: substitution generation  $\rightarrow_b$ , variable substitution  $\rightarrow_{\text{var}}$ , explicit garbage collection  $\rightarrow_{\text{gc}}$ , substitution propagation  $\rightarrow_{\text{prop}}$ . The first three rules are defined as expected. Substitution propagation is the set of rules which push explicit substitutions inside abstractions, applications, and other explicit substitutions.

As we work towards our method of simulating reduction, we will state the observations we make which influence our solution. We will also introduce our requirements of  $\lambda_z$  throughout the section as we encounter them. As  $\Lambda_{\text{sub}}$  has FCS, our first requirement is for simulation is:

**Requirement 9.1.** *The calculus  $\lambda_z$  must have full composition of substitutions.*

### 9.1.1 The problems

We want to simulate  $\Lambda_{\text{sub}}$  reduction in our second calculus so we define a translation between terms of the calculi. We start by taking the identity function on  $\Lambda x$  as our translation. Simulating  $\rightarrow_b$  and  $\rightarrow_{\text{gc}}$  should be straightforward so we try simulating  $\rightarrow_c$  reduction.

We will try to simulate the reduction  $t = (x (x x))[x/y] \rightarrow_c (x (x y))[x/y] \equiv u$  by finding a reduction path  $t \rightarrow_{\lambda_z} u$ . As we must replace a free occurrence of  $x$  with  $y$ , we try

$$(x (x x))[x/y] \rightarrow_{\text{prop}} (x[x/y] (x[x/y] x[x/y])) \rightarrow_{\text{var}} (x[x/y] (x[x/y] y)) = u'.$$

---

<sup>1</sup>This was Delia Kesner's suggestion.



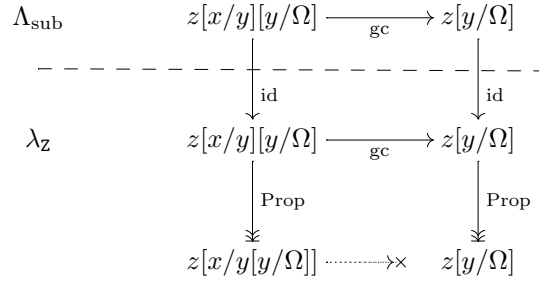
We encounter our first problem –  $u \neq u'$ ; the latter term has its explicit substitutions pushed inside. This was necessary to simulate the non-local  $\rightarrow_c$  substitution with the local  $\rightarrow_{\text{var}}$  substitution. To solve this problem, we need our translation to push some explicit substitutions inside the term. As we cannot decide which substitutions should be pushed inside, we define our translation as reduction to  $\rightarrow_{\text{Prop}}$  normal form where *all* substitutions are propagated as far as possible inside the term. Therefore, we require that  $\rightarrow_{\text{Prop}}$  has a normal form.

**Requirement 9.2.** *The propagation calculus  $\rightarrow_{\text{Prop}}$  of  $\lambda_Z$  must be strongly normalising.*

We make our first observation.

**Observation 9.1.** *To simulate non-local reduction with local reduction we need to compose our translation with reduction to  $\rightarrow_{\text{Prop}}$ -normal form.*

We have now solved the problem of simulating variable substitution. However, our solution introduces a new problem. Consider the reduction graph below.



If we do not use the normal form,  $\rightarrow_{\text{gc}}$  is easily simulated with the top square but if we do use it, then we cannot fill the outer square and simulate reduction. The problem now is that the outer substitution  $[y/\Omega]$  is pushed inside the inner one during the translation and the garbage collection discards both. We make another observation.

**Observation 9.2.** *To simulate garbage reduction our translation must not propagate explicit substitutions inside each other.*

Now we have a new problem – our two observations conflict with each other. For example, try simulating the reductions  $x[y/z][z/t] \rightarrow_c x[y/t][z/t]$  and  $x[y/z][z/t] \rightarrow_{\text{gc}} x[y/t]$ .

### 9.1.2 A solution

In order to simulate both  $\rightarrow_c$  and  $\rightarrow_{\text{gc}}$  in  $\lambda_Z$ , we need substitutions to be both pushed inside terms during the translation so that  $\rightarrow_c$  may be simulated, and also left outside so that they are not

unfairly garbage-collected. This is the classic problem of trying to be in two places at once! To solve this dilemma, we create *two* substitutions in our translation from  $\Lambda_{\text{sub}}$  to  $\lambda_z$  – one which may propagate through terms and one which must have restricted propagation.

The role of the first substitution is to propagate (and hence be copied) down into the term until a copy is at each free occurrence of the bound variable. Each copy is then ready to perform a substitution and thereby simulate  $\rightarrow_c$  reduction. We call this substitution (and its copies) a *mobile* substitution.

The second substitution is instead meant to be immobile; it waits up top, providing a syntactic match between a term and its translation. Our primary purpose for the simulation is to help prove normalisation properties of  $\Lambda_{\text{sub}}$ . If  $\lambda_z$  has FCS and PSN then explicit substitutions are probably propagated conditionally through terms. It is often the case that garbage substitutions are restricted so we therefore define this second substitution as garbage and call it (and its copies) an *idle* substitution.

This second substitution is meant to avoid the untimely garbage collection that led to Observation 9.2. However because we use  $\rightarrow_{\text{Prop}}$ -normal form in our translation, we must place a requirement on  $\lambda_z$ .

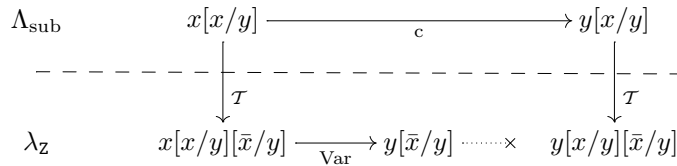
**Requirement 9.3.**  $\lambda_z$  must not compose the substitutions  $t[x/u][y/z]$  if  $y \notin \text{FV}(u)$ .

This seems quite reasonable<sup>2</sup>.

We have decided that our translation to  $\lambda_z$  will duplicate substitutions and then reduce to  $\rightarrow_{\text{Prop}}$ -normal form thereby allowing simulation of both  $\rightarrow_c$  and  $\rightarrow_{\text{gc}}$  reduction. However, there is one special case of  $\rightarrow_c$  which still causes trouble.

**Notation** ( $\bar{x}$ ). The notation  $\bar{x}$  denotes a variable which is fresh for some term.

Consider the reduction graph below.



<sup>2</sup>It is prudent at this point to ask whether this duplication of substitutions will interfere with strong normalisation, the property about which the simulation is meant to help us reason. Our intuition is as follows. The duplicate substitutions (the idle ones) are garbage. When they arise as subterms of strongly normalising terms, they *themselves* must be strongly normalising. They can conspire with substitutions *above* themselves to try and form infinite reduction sequences but then so can their mobile counterparts; this extra interaction only breaks normalisation if existing interactions involving mobile substitutions could. As garbage, they cannot affect any part of the term *below* them – they lie around waiting to be collected. Therefore, we believe that the duplication is entirely harmless from the point of view of strong normalisation.

Our translation has duplicated the  $\Lambda_{\text{sub}}$  substitution  $[x/y]$  with a garbage substitution  $[\bar{x}/y]$  where  $\bar{x}$  is a fresh variable. The problem here is that the  $\rightarrow_c$  reduction replaces the *last* free occurrence of a variable. However, while the translation of  $y[x/y]$  again duplicates the substitution, the mobile copy has been consumed by the  $\rightarrow_{\text{var}}$  reduction in  $\lambda_z$  and we cannot complete the square. We say that the  $\rightarrow_{\text{var}}$  reduction here is *garbage-collecting* since it automatically removes the now unnecessary substitution. This is generally the case.

**Observation 9.3.** *If the  $\rightarrow_{\text{var}}$  rule is garbage-collecting, then we define the translation<sup>3</sup> as:*

$$\begin{aligned} \mathcal{T}(t[x/u]) &:= \downarrow_{\text{Prop}}(\mathcal{T}(t)[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) && \text{if } x \in \text{FV}(t), \\ \mathcal{T}(t[x/u]) &:= \downarrow_{\text{Prop}}(\mathcal{T}(t)[x/\mathcal{T}(u)]) && \text{if } x \notin \text{FV}(t). \end{aligned}$$

If the  $\rightarrow_{\text{var}}$  rule is not garbage-collecting then we can probably always duplicate substitutions.

We now appear to have enough guidelines to define a translation into a calculus which satisfies our requirements. One question remains: when we try to simulate  $(\lambda x.t) u \rightarrow_b t[x/u]$ , at which stage do we create duplicate substitutions in  $\lambda_z$ ? We could alter the  $\rightarrow_b$  rule of  $\lambda_z$  to create an idle substitution as well as a mobile substitution. Alternatively, we could take a pessimistic approach and add idle substitutions to the translations of applications  $(t u)$  (or more particularly  $(\lambda x.t) u$ ). We now discuss the benefits and downsides to both approaches.

### 9.1.3 First approach: altering the target calculus

In the first approach, we alter the reduction relation of the target calculus  $\lambda_z$ . This is the approach we take in proving PSN of  $\Lambda_{\text{sub}}$  via simulation in  $\lambda_{\text{blxr}}$ , a modified  $\lambda_{\text{lxr}}$ . We reproduce this proof in Section 9.2.

We define the translation  $\mathcal{T}$  from  $\Lambda_{\text{sub}}$  to  $\lambda_z$  as follows.

$$\begin{aligned} \mathcal{T}(x) &:= x && \mathcal{T}(t[x/u]) &:= \downarrow_{\text{Prop}}(\mathcal{T}(t)[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) && \text{if } x \in \text{FV}(t) \\ \mathcal{T}(\lambda x.t) &:= \lambda x.\mathcal{T}(t) && \mathcal{T}(t[x/u]) &:= \downarrow_{\text{Prop}}(\mathcal{T}(t)[x/\mathcal{T}(u)]) && \text{if } x \notin \text{FV}(t) \\ \mathcal{T}(t u) &:= \mathcal{T}(t) \mathcal{T}(u) \end{aligned}$$

The translation of non-garbage substitutions adds a second substitution with a fresh binder  $\bar{x}$ . This substitution, the idle substitution, is always garbage.

We replace the  $\rightarrow_b$  rule of  $\lambda_z$  with a new rule which creates duplicate substitutions.

**Definition 9.1** ( $\rightarrow_{\text{bs}}$ ). *The reduction rule  $\rightarrow_{\text{bs}}$  (substitution/garbage generation) is defined as*

$$(\lambda x.t) u \rightarrow_{\text{bs}} t[x/u][\bar{x}/u], \text{ where } \bar{x} \text{ is fresh for } (\lambda x.t) u.$$

<sup>3</sup>The translation will get somewhat complicated; the translation makes copies of non-garbage substitutions. Therefore, translations of terms can grow exponentially with respect to the maximum nesting depth of non-garbage substitutions. However, we are only interested in proving a simulation so this is not of concern.

This allows us to match  $\Lambda_{\text{sub}}$  reduction sequences like  $(\lambda x.t)u \rightarrow_b t[x/u]$  with  $\lambda_z$  sequences  $(\lambda x.\mathcal{T}(t))\mathcal{T}(u) \rightarrow_{\text{bs}} \mathcal{T}(t)[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]$  followed by  $\downarrow_{\text{Prop}}$  where the idle substitution may be garbage-collected if  $x \notin \text{FV}(t)$ .

#### 9.1.4 Second approach: a pessimistic translation

In the second approach, we do not modify the target calculus but instead modify the translation  $\mathcal{T}$ . We call this new translation a *pessimistic* translation since we duplicate substitutions before time. The qualifier ‘pessimistic’ does not imply that this approach is worse than the first, just that we always assume that idle substitutions will be created. This approach was suggested by Delia Kesner and we use it in the proof of PSN for  $\Lambda_{\text{sub}}$  via simulation in **les** [81] (see Section 9.3).

The pessimistic translation  $\mathcal{T}$  from  $\Lambda_{\text{sub}}$  to  $\lambda_z$  is defined as follows.

$$\begin{aligned} \mathcal{T}(t[x/u]) &:= \downarrow_{\text{Prop}}(\mathcal{T}(t)[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) & \text{if } x \in \text{FV}(t) & \quad \mathcal{T}(x) &:= x \\ \mathcal{T}(t[x/u]) &:= \downarrow_{\text{Prop}}(\mathcal{T}(t)[x/\mathcal{T}(u)]) & \text{if } x \notin \text{FV}(t) & \quad \mathcal{T}(\lambda x.t) &:= \lambda x.\mathcal{T}(t) \\ \mathcal{T}(t \ u) &:= \downarrow_{\text{Prop}}((\mathcal{T}(t) \ \mathcal{T}(u))[\bar{x}/\mathcal{T}(u)]) \end{aligned}$$

This time, we prepare to match idle substitutions before they are created by duplicating the right hand side of applications as idle substitutions. This allows us to match  $\Lambda_{\text{sub}}$  reduction sequences  $(\lambda x.t)u \rightarrow_b t[x/u]$  with  $\lambda_z$  sequences  $(\lambda x.\mathcal{T}(t))\mathcal{T}(u)[\bar{x}/\mathcal{T}(u)] \rightarrow_b \mathcal{T}(t)[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]$  followed by  $\downarrow_{\text{Prop}}$  where the idle substitution may be garbage-collected if  $x \notin \text{FV}(t)$ . With this approach, the underlying calculus need not be changed.

#### 9.1.5 Comparison between the approaches

The first approach has an obvious flaw – PSN must be proven anew for the altered calculus. This may not present a significant problem as it may be possible to prove PSN using the original proof strategy; we prove PSN for  $\lambda\text{blxr}$  in the appendices using the proof for  $\lambda\text{lxr}$ . However, this requires extra work. On the other hand, the translation from  $\Lambda_{\text{sub}}$  to  $\lambda_z$  is the identity on pure terms. Thus, PSN for  $\Lambda_{\text{sub}}$  follows immediately from the simulation and PSN of the altered calculus.

The second approach does not alter the target calculus and thus avoids the extra work involved in re-proving PSN. On the other hand, we cannot immediately infer PSN for  $\Lambda_{\text{sub}}$  using the simulation as the translation is not the identity on pure terms. Our solution to this in Section 9.4 is to use a type system to infer normalisation properties of  $\Lambda_{\text{sub}}$  terms and their translations in  $\lambda_z^4$ . We explain that solution in more detail later but the idea is to use a type system where

<sup>4</sup>It may be possible to argue that if  $(t \ u)$  is strongly normalising then  $(t \ u)[\bar{x}/u], \bar{x} \notin \text{FV}(t \ u)$  is as well.

typable  $\Lambda$  terms are strongly normalising and where typing is preserved by the translation. These are not particularly strong requirements for a type system. We can infer from this that if a  $\Lambda_{\text{sub}}$  term is typable then it is strongly normalising *e.g.* see Theorem 9.51. This could be used, for example, in a proof that simply-typed  $\Lambda_{\text{sub}}$  terms are strongly normalising. If the type system also characterises  $\text{SN}_{\lambda_2}$  then we can infer PSN *e.g.* see Corollary 9.53. If the type system also satisfies subject reduction for  $\lambda_2$  then it is likely that the same type system will characterise  $\text{SN}_{\Lambda_{\text{sub}}}$  as the pessimistic translation is the identity (subject reduction allows us to ignore the  $\rightarrow_{\text{PROP}}$  reductions in the translation) except on two cases which add garbage, the body of which is typable.

Finally, the reader may note that the proof of PSN using  $\lambda\text{es}$  is significantly longer than the proof using  $\lambda\text{blxr}$ . This is misleading. The extra length is due to the use of induction in the proof with  $\lambda\text{es}$  which we find makes the proof much clearer and more immediately convincing.

## 9.2 PSN of $\Lambda_{\text{sub}}$ via simulation in $\lambda\text{blxr}$

To demonstrate the first approach of simulating non-local substitution with local substitution, we use an altered version of  $\lambda\text{lrx}$ . It satisfies Requirements 9.1 (FCS) and 9.3 (careful composition of substitutions). We first introduce the encoding of  $\Lambda_{\text{sub}}$  terms into the modified  $\lambda\text{lrx}$  in Section 9.2.1. In Section 9.2.2, we introduce a definition of the propagation calculus, showing that Requirement 9.2 (termination of propagation calculus) is satisfied, and the translation. We present the simulation and proof of PSN in Section 9.2.3.

### 9.2.1 The encoding of $\Lambda_{\text{sub}}$ in $\lambda\text{blxr}$

$\lambda\text{lrx}$  is not a conservative extension of the  $\lambda$ -calculus; we only consider *linear* terms. Its grammar also contains the explicit weakening and contraction term constructors. This means that we need to adapt our approach slightly. Our translation has to first encode non-linear  $\Lambda_{\text{sub}}$  terms into linear  $\lambda\text{lrx}$  terms. This is straightforward; we extend Kesner and Lengrand’s encoding of the  $\lambda$ -calculus, adding explicit substitutions and indices. We also need to deal with weakenings and contractions in the simulation and, as we show in the next section, we need to manipulate them in our translation.

**Definition 9.2** (Encoding of  $\Lambda_{\text{sub}}$  terms in  $\lambda\text{lrx}$ <sup>5</sup>). *The encoding  $\mathcal{A}$  of  $\Lambda_{\text{sub}}$  terms in  $\lambda\text{lrx}$  is an extension of the encoding of pure terms [80], indexed with a finite set which must contain all the*

<sup>5</sup>We previously mentioned a simpler encoding [122, Chapter 3.2]. Initial investigations suggested it insufficient for proving PSN via simulation but a suggestion was made which may overcome the issues. We have not explored this.

free variables of a  $\Lambda_{\text{sub}}$  term and none of its bound variables.

$$\begin{aligned}
\mathcal{A}(t)_X &:= W_{X \setminus \text{FV}(t)}(\mathcal{B}(t)) \\
\mathcal{B}(x) &:= x \\
\mathcal{B}(\lambda x.t) &:= \lambda x.\mathcal{A}(t)_{\text{FV}(t) \cup \{x\}} \\
\mathcal{B}(t \ u) &:= C_{\Phi}^{\Delta, \Pi} (R_{\Delta}^{\Phi}(\mathcal{B}(t)) \ R_{\Pi}^{\Phi}(\mathcal{B}(u))) \quad \text{where } \Phi := \text{FV}(t) \cap \text{FV}(u) \\
\mathcal{B}(t[x/u]) &:= C_{\Phi}^{\Delta, \Pi} ( (R_{\Delta}^{\Phi}(\mathcal{A}(t)_{\text{FV}(t) \cup \{x\}})) [x/R_{\Pi}^{\Phi}(\mathcal{B}(u))] ) \quad \text{where } x \notin \text{FV}(t) \\
\mathcal{B}(t[x/u]) &:= C_{\Phi}^{\Pi, \Psi_1} C_{\Theta \setminus \Phi}^{\Gamma_2, \Psi_2} (W_{\bar{x}} C_{\Pi}^{\Delta, \Gamma_1} (\mathcal{B}(R_{\Delta}^{\Phi}(t)) [x/u_1]) [\bar{x}/u_2]) \\
&\quad \text{where } x \in \text{FV}(t), u_1 = \mathcal{B}(R_{\Gamma_2}^{\Theta \setminus \Phi} R_{\Gamma_1}^{\Phi}(u)), u_2 = \mathcal{B}(R_{\Psi_2}^{\Theta \setminus \Phi} R_{\Psi_1}^{\Phi}(u))
\end{aligned}$$

where  $\Phi = (\text{FV}(t) \setminus \{x\}) \cap \text{FV}(u)$ ,  $\Theta = \text{FV}(u)$ , and  $\bar{x}$  is a fresh name in the last two cases. When an  $\mathcal{A}$ -encoding of a term  $t$  is not indexed,  $X$  is assumed to be  $\text{FV}(t)$  and when it is indexed by some undefined set  $X$ , it is assumed that  $\text{FV}(t) \subseteq X$ .

In our encoding,  $\Lambda_{\text{sub}}$  terms are indexed with a set  $X$  containing the free variables of the term. This was inspired by Milner's encoding of  $\Lambda_{\text{sub}}$  into  $\lambda$ BIG and is needed for our simulation as  $\Lambda_{\text{sub}}$  may lose free variables through reduction whereas  $\lambda$ lr does not. The requirement that the index set contains none of the bound variables of a term makes the encoding easier to specify and fits with the variable convention we have adopted. Note that the  $\mathcal{B}$ -encodings of an abstraction  $\lambda x.t$  or a closure  $t[x/u]$  use  $\mathcal{A}$  and will leave a weakening  $W_x$  directly below (up to  $\equiv_{C_w}$ ) the binder when  $x \notin \text{FV}(t)$ .

**Lemma 9.3** (properties of  $\mathcal{A}$ ).

1.  $\text{FV}(t) = \text{FV}(\mathcal{A}(t))$
2.  $\text{FV}(\mathcal{A}(t)_X) = X$
3.  $\mathcal{A}(R_{\Delta}^{\Phi}(t))_X = R_{\Delta}^{\Phi}(\mathcal{A}(t)_X)$

**Example.** The  $\{w, y, z\}$ -indexed encoding of  $((x \ x)y)[x/w \ y]$  is

$$W_z C_y^{y^1, y^3} C_w^{w_1, w_2} \left( W_{\bar{x}} (C_{y^1}^{y_1, y_2} ((C_x^{x_1, x_2} (x_1 \ x_2) y_1) [x/w_1 \ y_2])) [\bar{x}/w_2 \ y_3] \right).$$

The encodings of pure terms and terms  $t[x/u]$ ,  $x \notin \text{FV}(t)$  ensure linearity. The weakening  $W_x$  in the encoding of  $\lambda x.t$ ,  $x \notin \text{FV}(t)$  blocks any garbage arising from reducing the redex  $(\lambda x.t)u$  from propagating through the term. Similarly, the weakening  $W_x$  in the encoding of  $t[x/u]$ ,  $x \notin \text{FV}(t)$  blocks the propagation of garbage. The encoding of  $t[x/u]$ ,  $x \in \text{FV}(t)$  also ensures linearity and adds an idle substitution. The encodings of explicit substitutions were chosen to match reactums of substitution/garbage generation.

**Definition 9.4** ( $\rightarrow_{\text{bs}}$ ). The reduction rule  $\rightarrow_{\text{bs}}$  (substitution/garbage generation) is defined as

$$(\lambda x.t)u \rightarrow_{\text{bs}} C_{\Theta}^{\Gamma, \Psi}((W_{\bar{x}}(t[x/R_{\Gamma}^{\Theta}(u)]))[\bar{x}/R_{\Psi}^{\Theta}(u)])$$

where  $\Theta = \text{FV}(u)$  and  $\bar{x}$  is a fresh name.

This rule is a linear version of our generic definition.

**Terminology** (creates garbage). We say that a reduction  $(\lambda x.t)u \rightarrow_{\text{b}} t[x/u]'$  creates garbage when the abstraction in the redex binds a weakening  $W_x$ .

**Definition** ( $\lambda\text{blxr}$ ). We let  $\lambda\text{blxr}$  denote the calculus obtained from  $\lambda\text{lxr}$  by replacing  $\rightarrow_{\text{b}}$  with  $\rightarrow_{\text{bs}}$ . We let  $\rightarrow_{\lambda\text{blxr}}$  denote the reduction relation modulo congruence of  $\lambda\text{blxr}$ .

We are interested in proving PSN so it is crucial that our target calculus has this property.

**Proposition 9.5** (PSN for  $\lambda\text{blxr}$ ). For any pure term  $t$ , if  $t \in \text{SN}_{\beta}$  then  $\mathcal{B}(t) \in \text{SN}_{\lambda\text{blxr}}$ .

*Proof.* The encoding  $\mathcal{B}$  of Definition 9.2 is equivalent to the encoding  $\mathcal{B}$  of Definition B.30 on page 47 of Appendix B.3.3. This is shown with a proof by induction over the structure of  $\lambda$ -terms. The proof follows by Corollary B.48 on page 52 of Appendix B.3.  $\square$

**Corollary 9.6.** For any pure term  $t$ , if  $t \in \text{SN}_{\beta}$  then  $\mathcal{A}(t)_X \in \text{SN}_{\lambda\text{blxr}}$ .

*Proof.* Proof by induction on the cardinality of  $X - \text{FV}(t)$ . For the inductive step, note that if  $u$  is in  $\rightarrow_{\lambda\text{blxr}}$ -normal form then  $W_{\bar{z}}(u)$  is in  $\rightarrow_{\lambda\text{blxr}}$ -normal form.  $\square$

### 9.2.2 The propagation calculus for the simulation

The next step in our translation is to define the propagation calculus. Based on Observation 9.1, we will try to simulate the reduction sequence  $t = (x (x x))[x/y] \rightarrow_c (x (x y))[x/y] \equiv u$  in  $\lambda\text{lxr}$  by pushing substitutions inside the terms as far as possible. Assume that the union of the rules  $\text{App}_3, \text{App}_2, \text{Lamb}, \text{Comp}, \text{Weak}_2, \text{Cont}_1$  has a unique normal form represented by a function  $\downarrow_{\text{Psh}}$ . We define our translation as  $\mathcal{T} = \downarrow_{\text{Psh}} \circ \mathcal{A}$ . We start by replacing a free occurrence of  $x$  with  $y$ :

$$\begin{aligned} \mathcal{T}(t) &= \downarrow_{\text{Psh}} \circ C_y^{y l_1, y_4} \left( W_{\bar{x}} \left( C_x^{x_1, x l_1} (x_1 C_{x l_1}^{x_2, x_3} (x_2 x_3)) [x/y l_1] \right) [\bar{x}/y_4] \right) \\ &= C_y^{y l_1, y_4} \left( W_{\bar{x}} \left( C_{y l_1}^{y_1, y l_2} (x_1 [x_1/y_1] C_{y l_2}^{y_2, y_3} (x_2 [x_2/y_2] x_3 [x_3/y_3])) \right) [\bar{x}/y_4] \right) \\ &\rightarrow_{\text{Var}} C_y^{y l_1, y_4} \left( W_{\bar{x}} \left( C_{y l_1}^{y_1, y l_2} (x_1 [x_1/y_1] C_{y l_2}^{y_2, y_3} (x_2 [x_2/y_2] y_3)) \right) [\bar{x}/y_4] \right) \equiv t' \end{aligned}$$

However, the translation of  $u$  is

$$\begin{aligned}\mathcal{T}(u) &= \downarrow_{\text{Psh}} \circ C_y^{y^{l_1}, y^4} (W_{\bar{x}} (C_{y^{l_1}}^{y_3, y^{l_2}} (C_x^{x_1, x_2} (x_1 (x_2 y_3)) [x/y^{l_2}])) [\bar{x}/y_4]) \\ &= C_y^{y^{l_1}, y^4} (W_{\bar{x}} (C_{y^{l_1}}^{y_3, y^{l_2}} (C_{y^{l_2}}^{y_1, y^2} (x_1 [x_1/y_1] (x_2 [x_2/y_2] y_3)))) [\bar{x}/y_4]).\end{aligned}$$

This almost matches  $t'$  but the contractions are out of place. To fix this, we need to push the contractions inside as far as possible:

$$\begin{aligned}\mathcal{T}(u) &\equiv_{A, C_{1c}} C_y^{y^{l_1}, y^4} (W_{\bar{x}} (C_{y^{l_1}}^{y_1, y^{l_2}} (C_{y^{l_2}}^{y_2, y_3} (x_1 [x_1/y_1] (x_2 [x_2/y_2] y_3)))) [\bar{x}/y_4]) \\ &\rightarrow_{C_{\text{App}_1}} C_y^{y^{l_1}, y^4} (W_{\bar{x}} (C_{y^{l_1}}^{y_1, y^{l_2}} (x_1 [x_1/y_1] C_{y^{l_2}}^{y_2, y_3} (x_2 [x_2/y_2] y_3)))) [\bar{x}/y_4] \equiv t'\end{aligned}$$

So for  $\lambda$ l<sub>xr</sub>, we must push substitutions *and* contractions inside terms as far as possible. We therefore define the propagation calculus as a combination of these rules.

**Definition 9.7** ( $\rightarrow_{\text{Wk}}, \rightarrow_{\text{P}\downarrow c}$ ).

- $\rightarrow_{\text{Wk}}$  is defined as the union of **Weak**<sub>2</sub>, **WLamb**, **WApp**<sub>1</sub>, **WApp**<sub>2</sub>, **WSubs**, **Cross**, and **Merge**.
- $\rightarrow_{\text{Psh}}$  is defined as the union of **Lamb**, **App**<sub>3</sub>, **App**<sub>2</sub>, **Weak**<sub>2</sub>, **Cont**<sub>1</sub>, and **Comp**.
- $\rightarrow_{\text{Ctn}}$  is defined as the union of **CLamb**, **CApp**<sub>1</sub>, **CApp**<sub>2</sub>, **CSubs**, and **Cross**.
- $\rightarrow_{\text{P}\downarrow c}$ , the propagation calculus, is defined as the composition  $\rightarrow_{\text{Psh}} \rightarrow_{\text{Ctn}}$ .
- $\downarrow_{\text{P}\downarrow c}(t)$  is defined as the  $\rightarrow_{\text{P}\downarrow c}$ -normal form (up to  $\equiv$ ) of  $t$ .

**Definition 9.8** (translation  $\mathcal{T}$  from  $\Lambda_{\text{sub}}$  to  $\lambda$ bl<sub>xr</sub>). The translation  $\mathcal{T}$  from  $\Lambda_{\text{sub}}$  to  $\lambda$ bl<sub>xr</sub> is defined on  $\Lambda_{\text{sub}}$  terms  $t$  as  $\mathcal{T}(t)_X \stackrel{\text{def}}{=} \downarrow_{\text{P}\downarrow c}(\mathcal{A}(t)_X)$ . When the index  $X$  is omitted, it is assumed that  $X = \text{FV}(t)$ .

The relation  $\downarrow_{\text{P}\downarrow c}$  can be described as ‘push one substitution inside and then push all contractions in as far as possible.’ It has a unique normal form in at least all cases where we use it (Corollary 9.11).

Perhaps surprisingly (as it involves contractions), the **Merge** reduction rule is omitted in the normal form. This is because the encoding only introduces weakenings bound by an explicit substitution or abstraction but not contractions. Therefore, the rule cannot be applied to a  $\rightarrow_{\text{P}\downarrow c}$ -reduct of an encoding.

All branches of the abstract syntax tree of a  $\lambda$ l<sub>xr</sub> term occur at applications and substitutions. For any  $\lambda$ l<sub>xr</sub> term  $t$ ,  $\downarrow_{\text{P}\downarrow c}(t)$  has all substitutions lying directly above either a weakening  $W_x$  or a free occurrence of  $x$  and all contractions  $C_w^{y,z}$  pushed in as far as possible so that: 1) at the first



branching in the tree below the contraction, the left branch of the application (resp. substitution) below the contraction contains a free occurrence of  $y$  or  $z$  and the right branch contains a free occurrence of the other; or 2) both variables occur below the contraction and before any branch split<sup>6</sup>.

**Definition** (contractions at their most efficient). *We say that the contractions in a  $\lambda$ lrx term are at their most efficient when if  $x_1$  and  $x_2$  represent a variable  $x$ , there is a contraction  $C_{x_i}^{x_1, x_2}$  just above (up to congruence) the split in the abstract syntax tree where  $x_1$  is in one branch and  $x_2$  in the other.*

For example, consider  $C_x^{x_1, x_{l_1}}(y C_{x_{l_1}}^{x_2, x_3}((x_1 x_2) x_3))$ , the encoding of  $y ((x x) x)$ . This term does not have its contractions at their most efficient whereas its encoding  $\mathcal{A}(y ((x x) x)) \equiv y (C_x^{x_{l_1}, x_3}(C_{x_{l_1}}^{x_1, x_2}(x_1 x_2)) x_3)$  does. However, the former term can reduce to the latter.

**Proposition 9.9.** *Given a  $\Lambda_{\text{sub}}$  term  $t$ , any  $\rightarrow_{\text{plc}}$ -reduct  $t'$  of  $\mathcal{A}(t)_X$  has its contractions at their most efficient.*

*Proof.* Proof by induction over the length  $n$  of the  $\rightarrow_{\text{plc}}$  path from  $\mathcal{A}(t)_X$  to  $t'$  which is finite as  $\rightarrow_{\text{plc}} \subset \rightarrow_{\text{xr}}$  [122].  $\square$

**Corollary 9.10.** *For all  $\Lambda_{\text{sub}}$  terms  $t$ ,  $\mathcal{T}(t)_X$  is congruent to a term where the contractions are at their most efficient points.*

**Corollary 9.11.**  *$\mathcal{A}$ -images of  $\Lambda_{\text{sub}}$  terms have unique  $\rightarrow_{\text{plc}}$  normal forms (up to congruence).*

**Lemma 9.12.** *Let  $t$  be in  $\rightarrow_{\text{plc}}$ -normal form and let each weakening bound by an abstraction lie directly under (up to  $\equiv$ ) that abstraction. If  $t \rightarrow_{\lambda\text{blxr}} u$  is any reduction besides a  $\rightarrow_{\text{bs}}$  reduction which does not create garbage then  $u$  is in  $\rightarrow_{\text{plc}}$ -normal form.*

*Proof.* The proof is broken over the possible reductions, proving that the reductions do not create a  $\rightarrow_{\text{Psh}}$  redex [122].  $\square$

**Corollary 9.13.** *Let  $\mathcal{T}(t)_X \rightarrow_{\lambda\text{blxr}}^+ u$  for some  $\Lambda_{\text{sub}}$  term  $t$ . Unless the reduction sequence contains a  $\rightarrow_{\text{bs}}$  reduction which does not create garbage or a **CLamb** reduction,  $u$  is in  $\rightarrow_{\text{plc}}$ -normal form.*

*Proof.* It can be shown by induction that  $\mathcal{A}(t)_X$  has any weakening bound by an abstraction lying directly under that abstraction. This is also true of  $\mathcal{T}(t)_X$ . The result follows by Lemma 9.12 noting that only a **CLamb** reduction can come between a weakening and its binding abstraction (up to  $\equiv$ ) as **Lamb** redexes do not occur during the sequence.  $\square$

<sup>6</sup>In this second case, one of the free occurrences must be a weakening. Our encoding does not create such cases.

### 9.2.3 Proof of PSN using $\lambda$ blxr

Our proof uses labelled contexts for  $\Lambda_{\text{sub}}$  and  $\lambda$ blxr.

**Definition.** A labelled  $\Lambda_{\text{sub}}$  context  $C[\ ]_X$  is a term with a hole in it where  $\text{FV}([\ ]_X) = X$ . The term  $C[t]_X$  is defined as long as  $\text{FV}(t) = X$  by filling the hole with  $t$ . Similarly, a labelled  $\lambda$ blxr context  $C[\ ]_{\vec{X}}$  is a term with  $n$  holes which may be filled by a vector of  $n$  terms  $\vec{v} = v_1, \dots, v_n$  where  $v_i = R_{X_i}^{\text{FV}(t)}(v)$ ,  $X_i \cap X_j = \emptyset$  for  $1 \leq i \leq j \leq n$ .

We define  $\lambda$ blxr contexts in this way for our proof of PSN where each  $v_i$  will be a copy of an encoding of a  $\Lambda_{\text{sub}}$  redex, the copies being generated by encodings of non-garbage substitutions. The encoding  $\mathcal{A}$  can be extended as expected to labelled contexts [122].

Given a  $\Lambda_{\text{sub}}$  term  $C[t]_X$ , we have  $\mathcal{A}(C[t]_X) = \mathcal{A}(C)[t_1, t_2, \dots, t_n]_{\vec{X}}$  where  $t_i = \mathcal{A}(R_{X_i}^{\text{FV}(t)}(t))$  as the encoding may copy substitutions. In practice, we will only be interested in one copy of  $t$  and write  $\mathcal{A}(C)[t_i]$ , omitting the other copies and indexing.

The proof of PSN is quite complicated for a few reasons. The first is the syntax and linearity of  $\lambda$ blxr which complicates relatively simple terms. The second is our encoding which creates copies of substitutions, the number of which can be exponential with respect to the nesting level of the substitution with respect to other substitutions. A third reason is that our translation pushes substitutions through terms which creates even more copies (one per occurrence of the bound variable). Finally, to be frank, our use of labelled contexts creates more confusion. This last complication is avoided in the proof using  $\lambda$ es where we instead take an inductive approach.

To avoid introducing these complications into this chapter, we instead give a more abstract presentation of the proof, using reduction graphs. The brave reader is referred to Appendix B.4 (page 53) for the full proof. Our technical report [122] contains some examples of the cases.

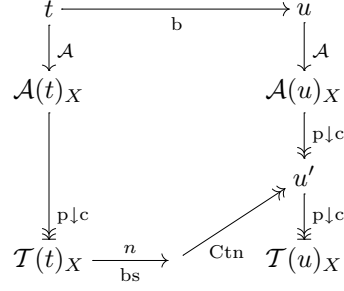
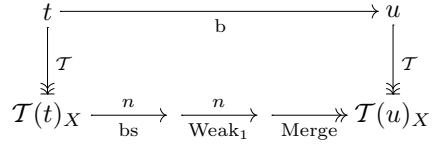
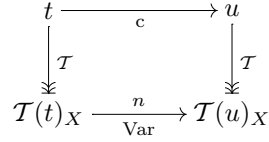
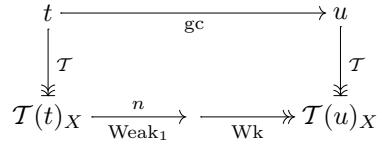
**Proposition 9.14.** *If  $t \rightarrow_{\text{bcgc}} u$  then  $\mathcal{T}(t)_X \rightarrow_{\lambda\text{blxr}}^+ \mathcal{T}(u)_X$ .*

*Proof.* Proof by case split. Figure 9.1 depicts the reduction graphs corresponding to the cases.  $\square$

**Corollary 9.15** (PSN for  $\Lambda_{\text{sub}}$ ).  $\rightarrow_{\text{bcgc}}$  preserves strong normalisation of  $\rightarrow_\beta$ .

*Proof.* Let  $t$  be any pure term which is strongly normalising for  $\rightarrow_\beta$ . By Corollary 9.6,  $\mathcal{T}(t)_X$  is strongly normalising for  $\rightarrow_{\lambda\text{blxr}}$ . By Proposition 9.14, any infinite  $\rightarrow_{\text{bcgc}}$  sequence starting from  $t$  induces an infinite reduction sequence starting from  $\mathcal{T}(t)_X$ . Hence, as  $\mathcal{T}(t)_X$  is strongly normalising, so is  $t$ .

Let  $t$  be any pure term which is not strongly normalising for  $\rightarrow_\beta$ . By Proposition 8.6.2, infinite  $\rightarrow_\beta$ -reductions induce infinite  $\rightarrow_{\text{bcgc}}$ -reductions.  $\square$

Reduction diagram for simulating  $\rightarrow_b$  which does not create garbageReduction diagram for simulating  $\rightarrow_b$  which does create garbageReduction diagram for simulating  $\rightarrow_c$ Reduction diagram for simulating  $\rightarrow_{gc}$ **Figure 9.1:** Reduction diagrams for simulating  $\Lambda_{\text{sub}}$  in  $\lambda\text{blxr}$

### 9.2.4 Sketch of proof of PSN by translation to $\Lambda_I$

We have proven PSN for  $\Lambda_{\text{sub}}$  using the fact that  $\lambda\text{blxr}$  has this property. The proof of PSN for  $\lambda\text{blxr}$  follows Lengrand’s approach of simulating reduction in  $\Lambda_I$  [93]. If we combine these simulations, we have a simulation of  $\rightarrow_{\text{bcgc}}$ -reduction in  $\Lambda_I$  as in Figure 9.2.

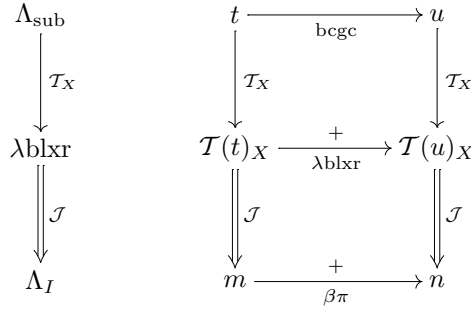


**Figure 9.2:** *Simulation of  $\rightarrow_{\text{bcgc}}$ -reduction in  $\Lambda_I$*

This immediately suggests that a proof of PSN for  $\Lambda_{\text{sub}}$  may be given by using a translation to  $\Lambda_I$  directly. The composition of  $\mathcal{T}$  and  $\mathcal{J}$  seems overkill as the translation duplicates substitutions and adds weakenings corresponding to the index set  $X$ .

Instead, we initially propose<sup>7</sup> the relation  $\mathcal{K}$  defined in Figure 9.3.  $\mathcal{K}$  relates  $\Lambda_{\text{sub}}$  terms with explicit substitutions to  $\lambda_I$  terms where the substitutions have been finished. The “memory operator” keeps track of garbage substitutions which are otherwise discarded in the relation. This is necessary in  $\Lambda_{\text{sub}}$  which does not have the linearity property (which  $\lambda\text{lrx}$  enjoys) that all binders bind a free occurrence of a variable.

$$\begin{array}{c}
\frac{}{x \mathcal{K} x} \qquad \frac{t \mathcal{K} m}{\lambda x.t \mathcal{K} \lambda x.m} x \in \text{FV}(t) \qquad \frac{t \mathcal{K} m}{\lambda x.t \mathcal{K} \lambda x.[m, x]} x \notin \text{FV}(t) \\
\\
\frac{t \mathcal{K} m \quad u \mathcal{K} n}{t u \mathcal{K} mn} \qquad \frac{t \mathcal{K} m}{t \mathcal{K} [m, p]} p \in \Lambda_I \\
\\
\frac{t \mathcal{K} m \quad u \mathcal{K} n}{t[x/u] \mathcal{K} t\{x \setminus n\}} x \in \text{FV}(t) \qquad \frac{t \mathcal{K} m \quad u \mathcal{K} n}{t[x/u] \mathcal{K} [t, n]} x \notin \text{FV}(t)
\end{array}$$

**Figure 9.3:** *Relating  $\Lambda_{\text{sub}}$  terms with  $\lambda_I$  terms*

<sup>7</sup>Stéphane Lengrand has since suggested a relation with less rules, replacing the condition  $x \in \text{FV}(t)$  in two of the rules with  $x \in \text{FV}(m)$  and removing the two rules with the condition  $x \notin \text{FV}(t)$ .

### 9.3 Simulating $\Lambda_{\text{sub}}$ reduction in $\lambda\text{es}$

To demonstrate the second approach of simulating non-local substitution with local substitution, we use  $\lambda\text{es}$ . This proof is simpler in many ways than our proof with  $\lambda\text{blxr}$ : we do not need to encode terms linearly; the reduction relation and syntax are much simpler and we do not need to modify the calculus and reprove PSN (the proof of PSN is presented in the following section). This is joint work with Delia Kesner.

$\lambda\text{es}$  satisfies Requirements 9.1 (FCS) and 9.3 (careful composition of substitutions). We take the relation  $\text{ALC}$  as our propagation calculus.

**Definition 9.16** ( $\rightarrow_{\text{ALC}}$ ).  $\rightarrow_{\text{ALC}}$  is defined as the union of  $\text{App}_1$ ,  $\text{App}_2$ ,  $\text{App}_3$ ,  $\text{Lamb}$ ,  $\text{Comp}_1$ , and  $\text{Comp}_2$  modulo the equivalence relation  $\text{E}_s$ .

The propagation calculus satisfies Requirement 9.2.

**Lemma 9.17.** *The ALC-normal forms of terms exist and are unique modulo  $\text{E}_s$ .*

*Proof.* The system  $\text{es}$  is terminating and so  $\text{ALC}$  is terminating and  $\text{ALC}$ -normal forms exist.  $\text{ALC}$  is locally confluent and locally coherent [81]. Therefore,  $\text{ALC}$  is confluent [75] and hence  $\text{ALC}$ -normal forms are unique modulo  $\text{ALC}$ -equivalence.  $\square$

We write  $t[\vec{x}/\vec{u}]$  for  $t[x_1/u_1] \dots [x_n/u_n]$  where  $\vec{x} = x_1, \dots, x_n$ ,  $\vec{u} = u_1, \dots, u_n$ , and  $x_i \notin \text{fv}(u_j)$  where  $i, j \in [1, \dots, n]$ . We write  $x_i$  to denote an arbitrary member of  $x_1, \dots, x_n$  and similarly for  $u_i$ . The concatenation of two vectors  $\vec{x}$  and  $\vec{y}$  is written as  $\vec{x}\vec{y}$ . We extend the set of variables with marked variables  $\bar{x}, \bar{y}$ , etc. denoting freshness.

**Definition 9.18** (translation  $\mathcal{T}$  from  $\Lambda_{\text{sub}}$  to  $\lambda\text{es}$ ). *The translation  $\mathcal{T}$  from  $\Lambda_{\text{sub}}$  to  $\text{ALC}$ -normal forms of  $\lambda\text{es}$  is defined as below, where  $\text{ALC}(t)$  denotes the reduction of  $t$  to  $\text{ALC}$ -normal form.*

$$\begin{aligned}
 \mathcal{T}(x) &= x \\
 \mathcal{T}(\lambda x.t) &= \lambda x.\mathcal{T}(t) \\
 \mathcal{T}(t \ u) &= (\mathcal{T}(t) \ \mathcal{T}(u))[\bar{z}/\mathcal{T}(u)] && \bar{z} \text{ is fresh} \\
 \mathcal{T}(t[y/u]) &= \text{ALC}(\mathcal{T}(t)[y/\mathcal{T}(u)]) && \text{if } y \notin \text{fv}(t) \\
 \mathcal{T}(t[y/u]) &= \text{ALC}(\mathcal{T}(t)[y/\mathcal{T}(u)][\bar{y}/\mathcal{T}(u)]) && \text{if } y \in \text{fv}(t) \text{ where } \bar{y} \text{ is fresh}
 \end{aligned}$$

In this approach, the translation of applications also introduces idle substitutions. In  $\lambda\text{es}$ , the propagation calculus may only propagate idle substitutions through abstractions<sup>8</sup> and not through applications or inside explicit substitutions.

<sup>8</sup>This seems unnecessary to us and disallowing such propagation may make our proof of simulation simpler.

**Lemma 9.19.**  $\text{fv}(t) = \text{fv}(\mathcal{T}(t))$

*Proof.* Induction over  $t$  with the fact that ALC reduction preserves free variables.  $\square$

**Lemma 9.20.** *If  $t$  is pure then:*

1.  $t = \downarrow_{\text{gc}}(\mathcal{T}(t))$ ;
2. *If  $t$  is in  $\rightarrow_{\Lambda_{\text{sub}}}$ -normal form then  $\downarrow_{\text{gc}}(\mathcal{T}(t))$  is in  $\rightarrow_{\lambda\text{es}}$ -normal form; and*
3. *If  $t$  is in  $\rightarrow_{\Lambda_{\text{sub}}}$ -normal form then  $\mathcal{T}(t) \in \text{SN}_{\lambda\text{es}}$ .*

*Proof.*

For (1), use induction on the structure of  $t$ . For (2), prove that  $\mathcal{T}(t)$  only has  $\rightarrow_{\text{gc}}$ -redexes by induction on the structure of  $t$ . As  $\rightarrow_{\text{gc}}$ -reductions do not create new redexes, any  $\rightarrow_{\text{gc}}$ -reduct of  $\mathcal{T}(t)$  has only  $\rightarrow_{\text{gc}}$ -redexes, if any. The result follows by  $\rightarrow_{\text{gc}}$  SN. (3) follows from (2).  $\square$

**Lemma 9.21.** *If  $t \rightarrow_{\lambda\text{es}} t'$  then  $t\{z/v\} \rightarrow_{\lambda\text{es}} t'\{z/v\}$ .*

*Proof.* Do a case split on  $t \rightarrow_{\lambda\text{es}} t'$  using induction on  $t$  when the redex is below the root [81].  $\square$

**Corollary 9.22.** *If  $t \rightarrow_{\lambda\text{es}}^+ t'$  then  $t\{z/v\} \rightarrow_{\lambda\text{es}}^+ t'\{z/v\}$ .*

In particular,  $t\{z/v\} \rightarrow_{\text{ALC}} \text{ALC}(t)\{z/v\}$ .

**Lemma 9.23.**  $\text{ALC}(t)\{z/\bar{z}[\bar{z}/v]\} = \text{ALC}(t[z/v])$  if  $z \in \text{fv}(t)$  and  $v$  in ALC-normal form.

*Proof.* Induction over the structure of  $t$  [81].  $\square$

**Lemma 9.24.** *If  $t \rightarrow_{\lambda\text{es}}^+ t'$  and  $t, t', v$  are in ALC-normal form then:*

1. *if  $z \notin \text{fv}(t)$  then  $\text{ALC}(t[z/v]) \rightarrow_{\lambda\text{es}}^+ \text{ALC}(t'[z/v])$ ;*
2. *if  $z \in \text{fv}(t)$  and  $z \in \text{fv}(t')$  then  $\text{ALC}(t[z/v]) \rightarrow_{\lambda\text{es}}^+ \text{ALC}(t'[z/v])$ ;*
3. *if  $z \in \text{fv}(t)$  and  $z \notin \text{fv}(t')$  then  $\text{ALC}(t[z/v]) \rightarrow_{\lambda\text{es}}^+ t'$ .*

*Proof.* Note that by our conventions,  $z \notin \text{bv}(t)$ .

1. Let  $z \notin \text{fv}(t)$ . Write  $t$  as  $\lambda\vec{y}.s$ , where  $s$  is not a  $\lambda$ -abstraction. As  $t \rightarrow_{\lambda\text{es}}^+ t'$  and  $t = \lambda\vec{y}.s$ , it must be that  $s \rightarrow_{\lambda\text{es}}^+ \lambda\vec{x}.s'$  with  $t' = \lambda\vec{y}.\lambda\vec{x}.s'$ , where  $s'$  is not a  $\lambda$ -abstraction and  $s, s'$  in ALC-normal form. Therefore,  $\text{ALC}(t[z/v]) = \lambda\vec{y}.s[z/v] \rightarrow_{\lambda\text{es}}^+ \lambda\vec{y}.\lambda\vec{x}.s'[z/v] \rightarrow_{\lambda\text{es}} \lambda\vec{y}.\lambda\vec{x}.s'[z/v] = \text{ALC}(t'[z/v])$ .

2/3. We define  $\mathcal{U}(p) = p\{z/\bar{z}[\bar{z}/v]\}$  for any  $p$  in the sequence  $t \rightarrow_{\lambda\text{es}}^+ t'$ . We can then build the following diagram

$$\begin{array}{ccccccc} t = t_1 & \xrightarrow{\lambda\text{es}} & t_2 & \xrightarrow{\lambda\text{es}} & \cdots & \xrightarrow{\lambda\text{es}} & t' \\ \downarrow \mathcal{U} & & \downarrow \mathcal{U} & & & & \downarrow \mathcal{U} \\ \mathcal{U}(t_1) & \xrightarrow{\lambda\text{es}} & \mathcal{U}(t_2) & \xrightarrow{\lambda\text{es}} & \cdots & \xrightarrow{\lambda\text{es}} & \mathcal{U}(t') \end{array}$$

where each square may be filled in by Lemma 9.21.

If  $z \in \text{fv}(t')$  then  $\mathcal{U}(t_1) = t\{z/\bar{z}[\bar{z}/v]\} = \text{ALC}(t[z/v]) \rightarrow_{\lambda\text{es}}^+ \mathcal{U}(t') = t'\{z/\bar{z}[\bar{z}/v]\} = \text{ALC}(t'[z/v])$ , using Lemma 9.23 twice.

If  $z \notin \text{fv}(t')$  then  $\mathcal{U}(t_1) = t\{z/\bar{z}[\bar{z}/v]\} = \text{ALC}(t[z/v]) \rightarrow_{\lambda\text{es}}^+ \mathcal{U}(t') = t'\{z/\bar{z}[\bar{z}/v]\} = t'$  using Lemma 9.23 once.

□

**Lemma 9.25.** *If  $u \rightarrow_{\lambda\text{es}}^+ u'$  and  $u$  is in ALC-normal form then  $\text{ALC}(t[z/u]) \rightarrow_{\lambda\text{es}}^+ \text{ALC}(t[z/u'])$ .*

*Proof.* The case  $z \notin \text{fv}(t)$  is straightforward;  $[z/u]$  may only be pushed through abstractions. For  $z \in \text{fv}(t)$ , use induction over the structure of  $t$  [81]. □

**Corollary 9.26.** *If  $u \rightarrow_{\lambda\text{es}}^+ u'$  and  $u$  is in ALC-normal form then  $\text{ALC}(t[z/u]) \rightarrow_{\lambda\text{es}}^+ \text{ALC}(t[z/u'])$ .*

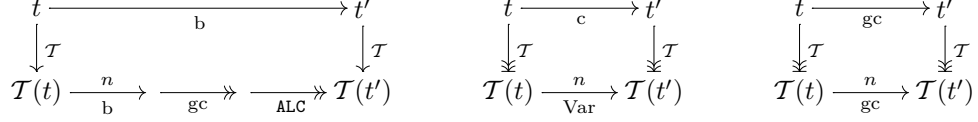
**Lemma 9.27.** *If  $\bar{x}$  is fresh for  $t$  and  $t'$  and  $\text{ALC}(t[\bar{x}/u]) \rightarrow_{\lambda\text{es}}^+ \text{ALC}(t'[\bar{x}/u])$  then  $\text{ALC}(t) \rightarrow_{\lambda\text{es}}^+ \text{ALC}(t')$ .*

*Proof.* The garbage substitution  $[\bar{x}/u]$  may not be copied during the sequence  $\text{ALC}(t[\bar{x}/u]) \rightarrow_{\lambda\text{es}}^+ \text{ALC}(t'[\bar{x}/u])$  and it is not discarded. It occurs exactly once in each term of the sequence. For any such term  $p$ , we define  $\mathcal{E}(p)$  as the term obtained by dropping the substitution  $[\bar{x}/u]$ .

The proof follows from the diagram below where the top line is the sequence  $\text{ALC}(t[\bar{x}/u]) \rightarrow_{\lambda\text{es}}^+ \text{ALC}(t'[\bar{x}/u])$ . The squares in the diagram can be completed as either i)  $t_i = \lambda\bar{z}.v_i[\bar{x}/u] \rightarrow_{\lambda\text{es}} \lambda\bar{z}.v_{i+1}[\bar{x}/u] = t_{i+1}$  with  $v_i \rightarrow_{\lambda\text{es}} v_{i+1}$  so that  $\mathcal{E}(t_i) = \lambda\bar{z}.v_i \rightarrow_{\lambda\text{es}} \lambda\bar{z}.v_{i+1}$ ; or ii)  $t_i = \lambda\bar{z}.(\lambda w.v_i)[\bar{x}/u] \rightarrow_{\lambda\text{es}} \lambda\bar{z}.\lambda w.v_i[\bar{x}/u] = t_{i+1}$  so that  $\mathcal{E}(t_i) = \lambda\bar{z}.\lambda w.v_i = \mathcal{E}(t_{i+1})$ .

$$\begin{array}{ccccccc} \lambda\bar{y}.\text{ALC}(v)[\bar{x}/u] = t_1 & \xrightarrow{\lambda\text{es}} & t_2 & \xrightarrow{\lambda\text{es}} & \cdots & \xrightarrow{\lambda\text{es}} & t_n \\ \downarrow \mathcal{E} & & \downarrow \mathcal{E} & & & & \downarrow \mathcal{E} \\ \lambda\bar{y}.\text{ALC}(v) = \mathcal{E}(t_1) & \xrightarrow[\lambda\text{es}]{*} & \mathcal{E}(t_2) & \xrightarrow[\lambda\text{es}]{*} & \cdots & \xrightarrow[\lambda\text{es}]{*} & \mathcal{E}(t_n) \end{array}$$

□

**Figure 9.4:** Reduction diagrams for simulating  $\Lambda_{\text{sub}}$  in  $\lambda\text{es}$ 

Again, we defer the details of the next proof to the appendix (page 61).

**Proposition 9.28** ( $\lambda\text{es}$  simulates  $\Lambda_{\text{sub}}$  through  $T$ ). *If  $t \rightarrow_{\Lambda_{\text{sub}}} t'$  then  $T(t) \rightarrow_{\lambda\text{es}}^+ T(t')$*

*Proof.* Proof by case split. Figure 9.4 depicts the reduction graphs corresponding to the cases.  $\square$

## 9.4 Characterisation of $\text{SN}_{\Lambda_{\text{sub}}}$ and a second proof of PSN

In this section, we present a characterisation of the strongly normalising terms of both  $\Lambda_{\text{sub}}$  and  $\lambda\text{es}$ . The characterisations use the same intersection type system which is an extension of the type system used to characterise the strongly normalising terms of the  $\lambda$ -calculus [40]. This is quite suggestive; we are able to show that a term of  $\Lambda_{\text{sub}}$  or  $\lambda\text{es}$  is strongly normalising if and only if it is a  $\rightarrow_b$ -reduct of a strongly normalising pure term indicating a close connection between normalisation properties of both explicit substitution calculi and the  $\lambda$ -calculus. This section is joint work with Delia Kesner.

### 9.4.1 Intuitions

To start off, we present our reasoning which lead to the characterisation of  $\text{SN}_{\Lambda_{\text{sub}}}$  and  $\text{SN}_{\lambda\text{es}}$  using intersection types. We begin with a pure additive type system and then try to add rules for explicit substitutions.

Assume we have an ES calculus  $\lambda_z$  with the set of terms  $\Lambda_x$ , the PSN property, and with  $\rightarrow_b$  as one of its reduction rules. As simply typed  $\lambda$ -terms are strongly normalising and  $\lambda_z$  has PSN, any pure term in  $\Lambda_x$  which is typable from the rules in Figure 2.6 is strongly normalising. Therefore, this type system characterises a subset of  $\text{SN}_{\lambda_z}$ .

We can extend the simply typed system to non-pure terms by adding the rule below (used by



Di Cosmo and Kesner to present simply typed  $\lambda\text{xgc}$  [51]).

$$\boxed{\frac{\Gamma \vdash u : B \quad \Gamma, x : B \vdash t : A}{\Gamma \vdash t[x/u] : A} \quad (\text{subs})}$$

This rule is quite natural – if we have  $(\lambda x.t)u \rightarrow_b t[x/u]$  and the redex is typed then we can type the reactum using the same environment by inspecting the final inference in the derivation<sup>9</sup>. The converse is also true *i.e.*  $\rightarrow_b$  satisfies subject reduction and expansion. We say that  $\rightarrow_b$  *preserves and reflects types* in this type system. We can use Herbelin’s approach [69] to prove that typable terms in this extended system are strongly normalising as follows.

**Definition 9.29** (C [78]). *The translation from  $\Lambda\text{x}$  terms to pure terms is:*

$$\begin{aligned} \mathbf{C}(x) &= x \\ \mathbf{C}(t \ u) &= \mathbf{C}(t) \ \mathbf{C}(u) \\ \mathbf{C}(\lambda x.t) &= \lambda x.\mathbf{C}(t) \\ \mathbf{C}(t[x/u]) &= (\lambda x.\mathbf{C}(t))\mathbf{C}(u). \end{aligned}$$

*i.e. the translation reverts all explicit substitutions to  $\beta$ -redexes.*

**Lemma 9.30.** *If  $t \in \Lambda\text{x}$  then  $t \twoheadrightarrow_b \mathbf{C}(t)$ .*

*Proof.* By induction over the structure of  $t$ . □

**Lemma 9.31.**  *$\mathbf{C}(t)$  is typable in the simply typed discipline iff  $t$  is typable in the extension with explicit substitutions.*

*Proof.* By induction over the structure of  $t$ . □

We conclude from PSN that terms typable in the extended type system are strongly normalising. We have now characterised a larger subset of  $\text{SN}_{\lambda_z}$ .

The intersection type discipline in Figure 2.7 is another extension of the pure simply typed system. Pottinger proved that this discipline characterises the strongly normalising terms of the  $\lambda$ -calculus [129]. We can play the same game as above and infer from PSN that a pure term in  $\Lambda\text{x}$  is typable from the rules in Figure 2.7 if and only if it is strongly normalising. We have now characterised an even larger (we can form a union with the previous subset) subset of  $\text{SN}_{\lambda_z}$  whose pure terms are exactly the set  $\text{SN}_{\beta}$ .

<sup>9</sup>The derivation is important; Curry’s system does not label binding variables with types.

Rather than forming this union, we can instead form a larger set<sup>10</sup> by extending the intersection type system with the rule **subs** for explicit substitutions. The complete set of rules is given in Figure 9.5. Once again, we combine Herbelin’s approach with PSN to infer that terms typable in the intersection type system with explicit substitutions are strongly normalising. Therefore, terms typable in this type system are strongly normalising for  $\Lambda_{\text{sub}}$ ,  $\lambda\text{es}$ , and  $\lambda\text{xgc}$ , as they all satisfy PSN. Further, we can show (similarly to Lemma 9.31) that all typable terms are reducts of strongly normalising pure terms.

Have we characterised the set of strongly normalising terms for these calculi? For  $\lambda\text{xgc}$ , we are not done. The term  $x[y/zz][z/\lambda v.vv]$  cannot be typed with this system – it is a reduct of the term  $(\lambda z.(\lambda y.x)(zz))(\lambda v.vv)$  which is not strongly normalising for the pure  $\lambda$ -calculus. However, this term is strongly normalising for  $\lambda\text{xgc}$  precisely because  $\lambda\text{xgc}$  does not have full composition of substitutions. But have we characterised  $\text{SN}_{\Lambda_{\text{sub}}}$  or  $\text{SN}_{\lambda\text{es}}$ ? We will turn our attention back to  $\lambda\text{xgc}$  first.

The type system in Figure 9.5 does not characterise  $\text{SN}_{\lambda\text{xgc}}$  but it does characterise a strict subset. Dougherty and Lescanne [53] demonstrated that the addition of an extra rule and a universal type characterises the set of terms which are normalising under leftmost reduction or head reduction in  $\lambda\text{xgc}$ . Lengrand et al. [94] point out that the intuition that the typing of a closure  $t[x/u]$  should follow from the typing of the  $\rightarrow_b$  redex  $(\lambda x.t)u$  – as in the **subs** rule above – fails for  $\lambda\text{xgc}$ . They state that “*we must allow the type system to distinguish between certain  $\rightarrow_b$ -redexes and their contractions.*” This is necessary in the example above where  $(\lambda z.(\lambda y.x)(zz))(\lambda v.vv)$  is not typable or strongly normalising but its  $\rightarrow_b$  reduct  $x[y/zz][z/\lambda v.vv]$  is strongly normalising. The lack of composition of substitutions in  $\lambda\text{xgc}$  is the cause.

Armed with this intuition, Lengrand et al. present a characterisation of  $\lambda\text{xgc}$  by extending the type system in Figure 9.5 with the two rules below.

|                            |                       |                         |         |
|----------------------------|-----------------------|-------------------------|---------|
| $\Delta \vdash u : B$      | $\Gamma \vdash t : A$ | $x \notin \text{av}(t)$ | (drop)  |
| $\Gamma \vdash t[x/u] : A$ |                       |                         |         |
| $\Delta \vdash u : B$      | $\Gamma \vdash t : A$ | $x \notin \Gamma$       | (K-cut) |
| $\Gamma \vdash t[x/u] : A$ |                       |                         |         |

The addition of these rules allows derivations which can essentially ignore garbage substitutions so long as they are typable *e.g.*  $x[y/zz][z/\lambda v.vv], x \neq y$  is now typable. A variable  $x$  is *available*

<sup>10</sup>The union does not include the term  $x[x/\lambda y.yy]$ .

in  $t$ , written  $x \in \text{av}(t)$ , if  $x \in \text{fv}(t)$  and a free occurrence exists which is not inside a garbage substitution *i.e.* a variable  $x$  is available if it could become available for substitution by  $[x/u]$  in some future reduct of  $t$ . It may not be available now since substitutions cannot be composed *e.g.* the occurrence of  $y$  in  $x[x/y][y/z]$  is available but cannot be replaced until the term reduces to  $y[y/z]$ .

This characterisation seems close to Rose's characterisation for  $\lambda\text{xgc}$ . A term is typable in this system only if all its bodies of substitution are typable (the **subs**, **K-cut**, and **drop** rules require this); this is similar in intention to Rose's definition of  $\text{subSN}$ . If  $x \notin \text{av}(t)$  then  $x \notin \downarrow_{\text{gc}}(t)$  so a term  $t[x/u]$  is typable only if  $\downarrow_{\text{gc}}(t)$  is typable.  $\downarrow_{\text{gc}}(t)$  contains no unavailable variables and no garbage. Therefore, its derivation does not use the **drop** or **K-cut** rules meaning that it is a reduct of a strongly normalising pure term and has a finite reduction path outside garbage *i.e.*  $\downarrow_{\text{gc}}(t)$  satisfies  $\# \text{gf} < \infty$ .

So how about  $\Lambda_{\text{sub}}$  and  $\lambda\text{es}$ ? The strongly normalising terms of these calculi form a strict subset of  $\text{SN}_{\lambda\text{xgc}}$  but they include the terms typable by (**axiom**), (**abs**), (**app**), ( $\wedge \text{I}$ ), ( $\wedge \text{E}$ ), and (**subs**). Do we need to add **drop** or **K-cut**?

If we naively add the rule **drop** then we can type the non-terminating term  $x[y/zz][z/\lambda v.vv]$ . However, the meaning of available variable differs in calculi with full composition. In these calculi, *the available variables are exactly the free variables* – they are available for substitution at any time. Using this definition of available variable, the rule **drop** is redundant. It allows the derivation  $\Gamma \vdash t[x/u] : A$  given  $\Delta \vdash u : B$  and  $\Gamma \vdash t : A$  where  $x \notin \text{FV}(t)$ . However, assume that  $t[x/u]$  is strongly normalising for  $\Lambda_{\text{sub}}$  and  $\lambda\text{es}$ . Then the pure term  $\mathcal{C}(t[x/u])$  is strongly normalising for  $\beta$  reduction (Lemma 9.48) and can be typed with the pure rules of Figure 9.5 and  $t[x/u]$  can be typed with the full set of rules. Therefore, the rule **drop** does not identify more strongly normalising terms.

The **K-cut** rule is still dangerous for  $\Lambda_{\text{sub}}$  and  $\lambda\text{es}$  because it allows the the derivation below.

$$\frac{\frac{\frac{}{x : A \vdash x : A} (\text{axiom})}{x : A \vdash x[y/zz] : A} (\text{K-cut}) \quad \frac{\dots}{\vdash \lambda v.vv : ((A \rightarrow A) \wedge A) \rightarrow A} (\text{K-cut})}{x : A \vdash x[y/zz][z/\lambda v.vv] : A} (\text{K-cut})$$

This is strong evidence that adding the **subs** rule to the intersection type system which characterises  $\text{SN}_{\beta}$  forms a type system which characterises both  $\Lambda_{\text{sub}}$  and  $\lambda\text{es}$ . Our intuition is that in a calculus with PSN and FCS, any term  $t$  which is strongly normalising is a reduct of a pure strongly normalising term  $\mathcal{C}(t)$ . Therefore, we consider the type systems in Figures 9.5 and 9.6 and prove in the next section that they do indeed yield a characterisation.

|                                                                                               |               |                                                                                         |               |
|-----------------------------------------------------------------------------------------------|---------------|-----------------------------------------------------------------------------------------|---------------|
| $\frac{}{\Gamma, x : A \vdash x : A}$                                                         | (axiom)       | $\frac{\Gamma \vdash u : B \quad \Gamma, x : B \vdash t : A}{\Gamma \vdash t[x/u] : A}$ | (subs)        |
| $\frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash (t u) : B}$ | (app)         | $\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \rightarrow B}$       | (abs)         |
| $\frac{\Gamma \vdash t : A \quad \Gamma \vdash t : B}{\Gamma \vdash t : A \wedge B}$          | ( $\wedge$ I) | $\frac{\Gamma \vdash t : A_1 \wedge A_2}{\Gamma \vdash t : A_i}$                        | ( $\wedge$ E) |

**Figure 9.5:** System  $\text{add}_{\lambda\text{S}}$ : An additive intersection type discipline for  $\lambda\text{es}$  and  $\Lambda_{\text{sub}}$ 

|                                                                                                       |                      |                                                                                                                             |                      |
|-------------------------------------------------------------------------------------------------------|----------------------|-----------------------------------------------------------------------------------------------------------------------------|----------------------|
| $\frac{}{x : A \vdash x : A}$                                                                         | (axiom)              | $\frac{\Gamma \vdash t : A \rightarrow B \quad \Delta \vdash u : A}{\Gamma \uplus \Delta \vdash (t u) : B}$                 | (app)                |
| $\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \rightarrow B}$                     | (abs <sub>1</sub> )  | $\frac{\Gamma \vdash t : B \text{ and } x \notin \Gamma}{\Gamma \vdash \lambda x. t : A \rightarrow B}$                     | (abs <sub>2</sub> )  |
| $\frac{\Gamma \vdash u : B \quad \Delta, x : B \vdash t : A}{\Gamma \uplus \Delta \vdash t[x/u] : A}$ | (subs <sub>1</sub> ) | $\frac{\Gamma \vdash u : B \quad \Delta \vdash t : A \text{ and } x \notin \Delta}{\Gamma \uplus \Delta \vdash t[x/u] : A}$ | (subs <sub>2</sub> ) |
| $\frac{\Gamma \vdash t : A \quad \Gamma \vdash t : B}{\Gamma \vdash t : A \wedge B}$                  | ( $\wedge$ I)        | $\frac{\Gamma \vdash t : A_1 \wedge A_2}{\Gamma \vdash t : A_i}$                                                            | ( $\wedge$ E)        |

**Figure 9.6:** System  $\text{mul}_{\lambda\text{S}}$ : A multiplicative intersection type discipline for  $\lambda\text{es}$  and  $\Lambda_{\text{sub}}$

Our proof strategy is as follows.

1. Prove that  $\mathcal{C}(t)$  is typable iff  $t$  is typable (Lemma 9.41) using the fact that  $\rightarrow_b$  preserves and reflects types.
2. Prove the square below (Lemma 9.47).

$$\begin{array}{ccc}
 \lambda\text{-calculus} & & \\
 & \begin{array}{ccc}
 \mathcal{C}(t) & \xrightarrow[\beta]{+} & t' \\
 \uparrow \mathcal{C} & & \uparrow \mathcal{C} \\
 t & \xrightarrow[\Lambda_{\text{sub}}]{+} & u
 \end{array} & \\
 \lambda_Z & & 
 \end{array}$$

3. Using the square, prove that  $t \in \text{SN}_{\lambda\text{es}}$  or  $t \in \text{SN}_{\Lambda_{\text{sub}}}$  implies  $t$  typable (Theorem 9.49).
4. Using PSN of  $\lambda\text{es}$ , prove that  $t$  typable implies  $t \in \text{SN}_{\lambda\text{es}}$  (Theorem 9.50).

This proves that the type systems characterise  $\lambda\text{es}$ . We now turn to  $\Lambda_{\text{sub}}$ .

1. Prove that the translation  $\mathcal{T}$  preserves typing (Lemma 9.44 and Corollary 9.45).
2. Using (1) and the simulation of  $\Lambda_{\text{sub}}$  reduction in  $\lambda\text{es}$ , conclude that  $t$  typable implies  $t \in \text{SN}_{\Lambda_{\text{sub}}}$  (Theorem 9.51).
3. Using (1), the characterisation of  $\text{SN}_{\lambda\text{es}}$  by typing, and PSN of  $\lambda\text{es}$ , conclude PSN for  $\Lambda_{\text{sub}}$ .

The simulation of  $\Lambda_{\text{sub}}$  reduction in  $\lambda\text{es}$  seems a useful tool for reasoning about properties which can be preserved or reflected through the translation  $\mathcal{T}$ . Here we consider the preservation of typing. In (2), the characterisation of  $\text{SN}_{\lambda\text{es}}$  is pulled back along the simulation to yield a characterisation of  $\text{SN}_{\Lambda_{\text{sub}}}$ . In (3), the proof of PSN is pulled back along the simulation using the characterisation of  $\text{SN}_{\lambda\text{es}}$ . These proofs are quite short; most of the work has been done in  $\lambda\text{es}$ .

## 9.4.2 Proof outline

We write  $\underline{n}$  for  $\{1, \dots, n\}$  and  $\wedge_{\underline{n}} A_i$  for  $A_1 \wedge \dots \wedge A_n$ .

The full proofs are available in the technical report [81]. For all the typing systems (Figures 2.6, 2.7, 9.5, and 9.6) we have the following lemmas.

**Lemma 9.32.** *If  $\Gamma \vdash t : A$  and  $A \ll B$ , then  $\Gamma \vdash t : B$ .*

*Proof.* By a straightforward induction on the definition of  $A \ll B$ . □

**Corollary 9.33.**  $\Gamma, x : \wedge_{\underline{n}} B_i \vdash x : B_j$  for all  $j \in \underline{n}$ .

**Lemma 9.34.** *If  $\Gamma, x : B \vdash t : A$  and  $C \ll B$ , then  $\Gamma, x : C \vdash t : A$ .*

*Proof.* By induction on  $\langle t, C \ll B, \Gamma, x : B \vdash t : A \rangle$ .  $\square$

**Lemma 9.35.** *Let  $\wedge_{\underline{n}} A_i \ll \wedge_{\underline{m}} B_j$ , where none of the  $A_i$  and  $B_j$  is an intersection type. Then for each  $B_j$  there is  $A_i$  s.t.  $B_j = A_i$ .*

*Proof.* By an induction on the definition of  $A \ll B$ .  $\square$

**Lemma 9.36.** *Let  $\Gamma$  be a context containing only  $\text{fv}(t)$ , then  $\Gamma \vdash_{\text{add}_{\lambda s}} t : A$  iff  $\Gamma, \Delta \vdash_{\text{add}_{\lambda s}} t : A$ .*

*Proof.* Both directions are proven by induction over the derivation of  $\Gamma \vdash_{\text{add}_{\lambda s}} t : A$ .  $\square$

We need generation lemmas for each system.

**Lemma 9.37** (Multiplicative Generation Lemma).

1.  $\Gamma \vdash x : A$  iff  $\Gamma = x : B$  and  $B \ll A$ .
2.  $\Gamma \vdash t u : A$  iff  $\Gamma = \Gamma_1 \uplus \Gamma_2$ , where  $\Gamma_1 = \text{fv}(t)$  and  $\Gamma_2 = \text{fv}(u)$  and there exist  $A_i, B_i, i \in \underline{n}$  s.t.  $\wedge_{\underline{n}} A_i \ll A$  and for all  $i \in \underline{n}$ ,  $\Gamma_1 \vdash t : B_i \rightarrow A_i$  and  $\Gamma_2 \vdash u : B_i$ .
3.  $\Gamma \vdash t[x/u] : A$  iff  $\Gamma = \Gamma_1 \uplus \Gamma_2$ , where  $\Gamma_1 = \text{fv}(t) \setminus \{x\}$  and  $\Gamma_2 = \text{fv}(u)$  and there exist  $A_i, B_i, i \in \underline{n}$  s.t.  $\wedge_{\underline{n}} A_i \ll A$  and for all  $i \in \underline{n}$ ,  $\Gamma_2 \vdash u : B_i$  and either  $x \notin \text{fv}(t)$  &  $\Gamma_1 \vdash t : A_i$  or  $x \in \text{fv}(t)$  &  $\Gamma_1, x : B_i \vdash t : A_i$ .
4.  $\Gamma \vdash \lambda x.t : A$  iff  $\Gamma = \text{fv}(\lambda x.t)$  and there exist  $A_i, B_i, i \in \underline{n}$  s.t.  $\wedge_{\underline{n}} (A_i \rightarrow B_i) \ll A$  and for all  $i \in \underline{n}$ , either  $x \notin \text{fv}(t)$  &  $\Gamma \vdash t : B_i$  or  $x \in \text{fv}(t)$  &  $\Gamma, x : A_i \vdash t : B_i$ .
5.  $\Gamma \vdash \lambda x.t : B \rightarrow C$  iff  $\Gamma = \text{fv}(\lambda x.t)$  and  $\Gamma, x : B \vdash t : C$  or  $\Gamma \vdash t : C$ .

*Proof.* The right to left implications follow from the typing rules in Figures 2.8 and 9.6 and Lemmas 9.34 and 9.32.

The left to right implication of the first four points are shown by induction on the typing derivation of the left part. The left to right implication of point 5 follows from point 4 and Lemma 9.35.  $\square$

**Lemma 9.38** (Additive Generation Lemma).

1.  $\Gamma \vdash x : A$  iff there is  $x : B \in \Gamma$  and  $B \ll A$ .
2.  $\Gamma \vdash t u : A$  iff there exist  $A_i, B_i, i \in \underline{n}$  s.t.  $\wedge_{\underline{n}} A_i \ll A$  and  $\Gamma \vdash t : B_i \rightarrow A_i$  and  $\Gamma \vdash u : B_i$ .

3.  $\Gamma \vdash t[x/u] : A$  iff there exist  $A_i, B_i, i \in \underline{n}$  s.t.  $\bigwedge_{\underline{n}} A_i \ll A$  and for all  $i \in \underline{n}$ ,  $\Gamma \vdash u : B_i$  and  $\Gamma, x : B_i \vdash t : A_i$ .
4.  $\Gamma \vdash \lambda x.t : A$  iff there exist  $A_i, B_i, i \in \underline{n}$  s.t.  $\bigwedge_{\underline{n}} (A_i \rightarrow B_i) \ll A$  and for all  $i \in \underline{n}$ ,  $\Gamma, x : A_i \vdash t : B_i$ .
5.  $\Gamma \vdash \lambda x.t : B \rightarrow C$  iff  $\Gamma, x : B \vdash t : C$ .

*Proof.* The right to left implications follow from the typing rules in Figure 2.7 and 9.5 and Lemmas 9.34 and 9.32.

The left to right implication of the first four points are shown by induction on the typing derivation of the left part. The left to right implication of point 5 follows from point 4 and Lemma 9.35.  $\square$

There is a correspondence between the multiplicative and additive systems.

**Lemma 9.39.** *Let  $t$  be a  $\Lambda\text{x}$ -term. Then  $\Gamma \vdash_{\text{add}_{\lambda_s}} t : A$  iff  $\Gamma \wedge \text{fv}(t) \vdash_{\text{mul}_{\lambda_s}} t : A$ . Moreover, if  $t$  is a pure  $\lambda$ -term, then  $\Gamma \vdash_{\text{add}_{\lambda}} t : A$  iff  $\Gamma \wedge \text{fv}(t) \vdash_{\text{mul}_{\lambda}} t : A$ .*

*Proof.* The right to left implication is by induction on  $t$  using both generation lemmas and Lemma 9.36. The left to right implication is by induction on  $t$  using the generation lemmas.  $\square$

**Lemma 9.40.** *Let  $t$  be a  $\lambda$ -term. Then  $\Gamma \vdash_{\text{add}_{\lambda}} t : A$  iff  $\Gamma \vdash_{\text{add}_{\lambda_s}} t : A$ .*

*Proof.* By induction on  $t$  using the generation lemma 9.38.  $\square$

**Lemma 9.41.** *Let  $t$  be a  $\Lambda\text{x}$ -term. Then  $\Gamma \vdash_{\text{add}_{\lambda}} \mathcal{C}(t) : A$  iff  $\Gamma \vdash_{\text{add}_{\lambda_s}} t : A$ .*

*Proof.* By induction on  $t$  using the generation lemma 9.38.  $\square$

**Lemma 9.42.**  $\mathcal{C}(t)\{x/\mathcal{C}(u)\} = \mathcal{C}(t\{x/u\})$ .

*Proof.* By induction on  $t$ .  $\square$

**Lemma 9.43** (Partial Subject Reduction). *If  $\Gamma \vdash_{\text{add}_{\lambda_s}} t : A$  then  $\Gamma \vdash_{\text{add}_{\lambda_s}} \text{ALC}(t) : A$ .*

*Proof.* By Lemma 9.39,  $\Gamma \wedge \text{fv}(t) \vdash_{\text{mul}_{\lambda_s}} t : A$ . By Subject Reduction [78],  $\Gamma \wedge \text{fv}(t) \vdash_{\text{mul}_{\lambda_s}} \text{ALC}(t) : A$ . Since  $\text{fv}(t) = \text{fv}(\text{ALC}(t))$ , we conclude  $\Gamma \vdash_{\text{add}_{\lambda_s}} \text{ALC}(t) : A$  by Lemma 9.39.  $\square$

**Lemma 9.44.** *If  $\Gamma \vdash_{\text{mul}_{\lambda_s}} t : A$ , then  $\Gamma \vdash_{\text{mul}_{\lambda_s}} \mathcal{T}(t) : A$ .*

*Proof.* By induction on  $t$  using the Generation Lemma 9.37.

- If  $t = x$ , then the property is straightforward.
- If  $t = \lambda x.u$ , then the property holds by the i.h.
- If  $t = u v$ , then  $\Gamma = \Gamma_1 \uplus \Gamma_2$ , where  $\Gamma_1 = \text{fv}(u)$  and  $\Gamma_2 = \text{fv}(v)$  and there exist  $A_i, B_i, i \in \underline{n}$  s.t.  $\bigwedge_{\underline{n}} A_i \ll A$  and for all  $i \in \underline{n}$ ,  $\Gamma_1 \vdash u : B_i \rightarrow A_i$  and  $\Gamma_2 \vdash v : B_i$ . By the i.h.  $\Gamma_1 \vdash \mathcal{T}(u) : B_i \rightarrow A_i$  and  $\Gamma_2 \vdash \mathcal{T}(v) : B_i$ . Then  $\Gamma \vdash \mathcal{T}(u) \mathcal{T}(v) : A_i$ . For fresh  $z$ ,  $\Gamma \vdash (\mathcal{T}(u) \mathcal{T}(v))[z/\mathcal{T}(v)] : A_i$ . Thus  $\Gamma \vdash (\mathcal{T}(u) \mathcal{T}(v))[z/\mathcal{T}(v)] : A$ .
- If  $t = u[x/v], x \notin \text{fv}(u)$ , then  $\Gamma = \Gamma_1 \uplus \Gamma_2$ , where  $\Gamma_1 = \text{fv}(u) \setminus \{x\}$  and  $\Gamma_2 = \text{fv}(v)$  and there exist  $A_i, B_i, i \in \underline{n}$  s.t.  $\bigwedge_{\underline{n}} A_i \ll A$  and for all  $i \in \underline{n}$ ,  $\Gamma_2 \vdash v : B_i$  and  $\Gamma_1 \vdash u : A_i$ . By the i.h.  $\Gamma_2 \vdash \mathcal{T}(v) : B_i$  and  $\Gamma_1 \vdash \mathcal{T}(u) : A_i$ . Thus  $\Gamma \vdash \mathcal{T}(u)[x/\mathcal{T}(v)] : A_i$ . By Lemma 9.43, we conclude  $\Gamma \vdash \text{ALC}(\mathcal{T}(u)[x/\mathcal{T}(v)]) : A_i$ . Thus,  $\Gamma \vdash \text{ALC}(\mathcal{T}(u)[x/\mathcal{T}(v)]) : A$ .
- If  $t = u[x/v], x \in \text{fv}(u)$ , then  $\Gamma = \Gamma_1 \uplus \Gamma_2$ , where  $\Gamma_1 = \text{fv}(u) \setminus \{x\}$  and  $\Gamma_2 = \text{fv}(v)$  and there exist  $A_i, B_i, i \in \underline{n}$  s.t.  $\bigwedge_{\underline{n}} A_i \ll A$  and for all  $i \in \underline{n}$ ,  $\Gamma_2 \vdash v : B_i$  and  $\Gamma_1, x : B_i \vdash u : A_i$ . By the i.h.  $\Gamma_2 \vdash \mathcal{T}(v) : B_i$  and  $\Gamma_1, x : B_i \vdash \mathcal{T}(u) : A_i$ . Thus  $\Gamma \vdash \mathcal{T}(u)[x/\mathcal{T}(v)] : A_i$ . For fresh  $\bar{x}$ ,  $\Gamma \vdash \mathcal{T}(u)[x/\mathcal{T}(v)][\bar{x}/\mathcal{T}(v)] : A_i$ . By Lemma 9.43, we conclude  $\Gamma \vdash \text{ALC}(\mathcal{T}(u)[x/\mathcal{T}(v)][\bar{x}/\mathcal{T}(v)]) : A_i$ . Thus,  $\Gamma \vdash \text{ALC}(\mathcal{T}(u)[x/\mathcal{T}(v)][\bar{x}/\mathcal{T}(v)]) : A$ .

□

**Corollary 9.45.** *If  $\Gamma \vdash_{\text{add}_{\lambda\text{s}}} t : A$ , then  $\Gamma \vdash_{\text{add}_{\lambda\text{s}}} \mathcal{T}(t) : A$ .*

*Proof.* Follows from Lemmas 9.44 and 9.39. □

**Lemma 9.46** (Full Composition).  *$t[x/u] \rightarrow_{\lambda\text{es}}^+ t\{x/u\}$  and  $t[x/u] \rightarrow_{\Lambda_{\text{sub}}}^+ t\{x/u\}$ .*

*Proof.* Proven previously for  $\lambda\text{es}$  [78] and  $\Lambda_{\text{sub}}$  [122] (see also Section 7.3.7). □

**Lemma 9.47.** *If  $\mathcal{C}(t) \rightarrow_{\beta} t'$ , then there is  $u$  s.t.  $t \rightarrow_{\lambda\text{es}}^+ u$  (resp.  $t \rightarrow_{\Lambda_{\text{sub}}}^+ u$ ) and  $t' = \mathcal{C}(u)$ .*

*Proof.* We reason by induction on the reduction step  $\mathcal{C}(t) \rightarrow_{\beta} t'$  to show that  $t \rightarrow_{\lambda\text{es}}^+ u$ . The case  $t \rightarrow_{\Lambda_{\text{sub}}}^+ u$  is similar.

If the step is external, then we have two possibilities.

- If  $\mathcal{C}((\lambda x.t_1) t_2) = (\lambda x.\mathcal{C}(t_1)) \mathcal{C}(t_2) \rightarrow_{\beta} \mathcal{C}(t_1)\{x/\mathcal{C}(t_2)\}$ , then  $(\lambda x.t_1) t_2 \rightarrow_{\beta} t_1[x/t_2] \rightarrow_{\lambda\text{es}}^+ t_1\{x/t_2\}$  by Lemma 9.46. We conclude by Lemma 9.42.
- If  $\mathcal{C}(t_1[x/t_2]) = (\lambda x.\mathcal{C}(t_1)) \mathcal{C}(t_2) \rightarrow_{\beta} \mathcal{C}(t_1)\{x/\mathcal{C}(t_2)\}$ , then  $t_1[x/t_2] \rightarrow_{\lambda\text{es}}^+ t_1\{x/t_2\}$  by Lemma 9.46. We conclude again by Lemma 9.42.



If the step is internal, then we reason by cases.

- If  $\mathcal{C}(t_1 \ t_2) = \mathcal{C}(t_1) \ \mathcal{C}(t_2) \rightarrow_{\beta} t'_1 \ \mathcal{C}(t_2)$ , then  $t_1 \rightarrow_{\lambda_{\text{ES}}}^+ u_1$  and  $t'_1 = \mathcal{C}(u_1)$  by the i.h. so that  $t_1 \ t_2 \rightarrow_{\lambda_{\text{ES}}}^+ u_1 \ t_2$  and  $t'_1 \ \mathcal{C}(t_2) = \mathcal{C}(u_1 \ t_2)$ .
- If  $\mathcal{C}(t_1 \ t_2) = \mathcal{C}(t_1) \ \mathcal{C}(t_2) \rightarrow_{\beta} \mathcal{C}(t'_1) \ t'_2$ , then this case is similar to the previous one.
- If  $\mathcal{C}(\lambda x.t_1) = \lambda x.\mathcal{C}(t_1) \rightarrow_{\beta} \lambda x.t'_1$ , then  $t_1 \rightarrow_{\lambda_{\text{ES}}}^+ u_1$  and  $t'_1 = \mathcal{C}(u_1)$  by the i.h. so that  $\lambda x.t_1 \rightarrow_{\lambda_{\text{ES}}}^+ \lambda x.u_1$  and  $\lambda x.t'_1 = \mathcal{C}(\lambda x.u_1)$ .
- If  $\mathcal{C}(t_1[x/t_2]) = (\lambda x.\mathcal{C}(t_1)) \ \mathcal{C}(t_2) \rightarrow_{\beta} (\lambda x.t'_1) \ \mathcal{C}(t_2)$ , then  $t_1 \rightarrow_{\lambda_{\text{ES}}}^+ u_1$  and  $t'_1 = \mathcal{C}(u_1)$  by the i.h. so that  $t_1[x/t_2] \rightarrow_{\lambda_{\text{ES}}}^+ u_1[x/t_2]$  and  $(\lambda x.t'_1) \ \mathcal{C}(t_2) = \mathcal{C}(u_1[x/t_2])$ .
- If  $\mathcal{C}(t_1[x/t_2]) = (\lambda x.\mathcal{C}(t_1)) \ \mathcal{C}(t_2) \rightarrow_{\beta} (\lambda x.\mathcal{C}(t_1)) \ t'_2$ , then this case is similar to the previous one.

□

**Lemma 9.48.** *If  $t \in \text{SN}_{\lambda_{\text{ES}}}$  (resp.  $t \in \text{SN}_{\Lambda_{\text{SUB}}}$ ) then  $\mathcal{C}(t) \in \text{SN}_{\beta}$ .*

*Proof.* Suppose  $\mathcal{C}(t) \notin \text{SN}_{\beta}$ . Thus, there is an infinite  $\beta$ -reduction sequence starting at  $\mathcal{C}(t)$ , i.e.,

$$\mathcal{C}(t) \rightarrow_{\beta} t_1 \rightarrow_{\beta} t_2 \rightarrow_{\beta} \dots$$

By Lemma 9.47 we can construct infinite reduction sequences

$$t \rightarrow_{\lambda_{\text{ES}}}^+ u_1 \rightarrow_{\lambda_{\text{ES}}}^+ u_2 \rightarrow_{\lambda_{\text{ES}}}^+ \dots \quad \text{and} \quad t \rightarrow_{\Lambda_{\text{SUB}}}^+ v_1 \rightarrow_{\Lambda_{\text{SUB}}}^+ v_2 \rightarrow_{\Lambda_{\text{SUB}}}^+ \dots$$

s.t.  $t_i = \mathcal{C}(u_i)$  and  $t_i = \mathcal{C}(v_i)$  for all  $i$ . Then  $t \notin \text{SN}_{\lambda_{\text{ES}}}$  and  $t \notin \text{SN}_{\Lambda_{\text{SUB}}}$  which is a contradiction. □

**Theorem 9.49.** *If  $t \in \text{SN}_{\lambda_{\text{ES}}}$  (resp.  $t \in \text{SN}_{\Lambda_{\text{SUB}}}$ ), then  $t$  is typable in the additive and multiplicative systems  $\text{add}_{\lambda_{\text{S}}}$  and  $\text{mul}_{\lambda_{\text{S}}}$ .*

*Proof.* Let  $t \in \text{SN}_{\lambda_{\text{ES}}}$  (resp.  $t \in \text{SN}_{\Lambda_{\text{SUB}}}$ ). Then  $\mathcal{C}(t) \in \text{SN}_{\beta}$  by Lemma 9.48 and  $\mathcal{C}(t)$  is typable in  $\text{add}_{\lambda}$  [129]. By Lemma 9.41,  $t$  is typable in  $\text{add}_{\lambda_{\text{S}}}$ . By Lemma 9.39,  $t$  is typable in  $\text{mul}_{\lambda_{\text{S}}}$ . □

**Theorem 9.50** (SN for  $\lambda_{\text{ES}}$ ). *If  $t$  is typable in the additive or multiplicative systems  $\text{add}_{\lambda_{\text{S}}}$  and  $\text{mul}_{\lambda_{\text{S}}}$ , then  $t \in \text{SN}_{\lambda_{\text{ES}}}$ .*

*Proof.* If  $t$  is typable in  $\text{mul}_{\lambda_{\text{S}}}$ , then  $t$  is also typable in  $\text{add}_{\lambda_{\text{S}}}$  by Lemma 9.39 so assume directly that  $\Gamma \vdash_{\text{add}_{\lambda_{\text{S}}}} t : A$ . We have  $\mathcal{C}(t) \rightarrow_{\lambda_{\text{ES}}}^* t$  and Lemma 9.41 gives  $\mathcal{C}(t)$  typable in  $\text{add}_{\lambda}$ . Thus  $\mathcal{C}(t)$  is  $\beta$ -strongly normalising [129]. As a consequence,  $\mathcal{C}(t) \in \text{SN}_{\lambda_{\text{ES}}}$  by PSN [78]. Thus  $t \in \text{SN}_{\lambda_{\text{ES}}}$ . □

**Theorem 9.51** (SN for  $\Lambda_{sub}$ ). *If  $t$  is typable in the additive or multiplicative systems  $\mathbf{add}_{\lambda_s}$  and  $\mathbf{mul}_{\lambda_s}$ , then  $t \in \mathbf{SN}_{\Lambda_{sub}}$ .*

*Proof.* We can assume directly that  $\Gamma \vdash_{\mathbf{mul}_{\lambda_s}} t : A$  by Lemma 9.39. By Lemma 9.44,  $T(t)$  is typable in  $\mathbf{mul}_{\lambda_s}$ . Thus,  $T(t) \in \mathbf{SN}_{\lambda_{es}}$  by Theorem 9.50. Suppose  $t \notin \mathbf{SN}_{\Lambda_{sub}}$ . Given an infinite  $\Lambda_{sub}$ -reduction sequence starting at  $t$  we can construct, by Lemma 9.28, an infinite  $\lambda_{es}$ -reduction sequence starting at  $T(t)$ . This leads to a contradiction.  $\square$

**Corollary 9.52.**  *$t$  is typable in  $\mathbf{add}_{\lambda_s}$  iff  $t$  is typable in  $\mathbf{mul}_{\lambda_s}$  iff  $t \in \mathbf{SN}_{\lambda_{es}}$  iff  $t \in \mathbf{SN}_{\Lambda_{sub}}$ .*

**Corollary 9.53** (PSN for  $\Lambda_{sub}$ ).  *$t \in \mathbf{SN}_{\beta}$  if and only if  $t \in \mathbf{SN}_{\Lambda_{sub}}$ .*

*Proof.* If  $t \in \mathbf{SN}_{\beta}$ , then  $t \in \mathbf{SN}_{\lambda_{es}}$  by PSN of  $\lambda_{es}$  [78], so that  $t$  is typable by Theorem 9.49 and  $T(t)$  is typable by Lemma 9.44. Theorem 9.50 gives  $T(t) \in \mathbf{SN}_{\lambda_{es}}$ . Now, suppose  $t \notin \mathbf{SN}_{\Lambda_{sub}}$ . Given an infinite  $\Lambda_{sub}$ -reduction sequence from  $t$  we can construct, by Lemma 9.28, an infinite  $\lambda_{es}$ -reduction sequence from  $T(t)$ . This leads to a contradiction, thus  $t \in \mathbf{SN}_{\Lambda_{sub}}$ .

If  $t \notin \mathbf{SN}_{\beta}$  then Proposition 8.6 proves that  $t \notin \mathbf{SN}_{\Lambda_{sub}}$ .  $\square$

## 9.5 Conclusions

We explained the problems involved with simulating reduction of a calculus with non-local explicit substitution (the source calculus) in a calculus with local explicit substitution (the target calculus). We identified some simple requirements for the target calculus and presented a novel idea for constructing a simulation using mobile and idle copies of substitutions.

Two approaches were discussed using different translations from source to target. The first altered the target calculus to create an idle copy during  $\rightarrow_b$  reduction. The downside to this approach is that if the simulation is to be used for a proof of PSN for the source calculus, that property must be proven anew for the altered target calculus. However, we do not consider the alteration dangerous and propose that the proof of PSN for the unaltered target calculus may sometimes be adapted to allow a new proof. To support this claim, we present a proof of PSN for  $\lambda_{blxr}$  in Appendix B.3. The upside to this approach is that once the simulation is proven, PSN follows easily as the translation is the identity on pure terms.

The second approach left the target calculus intact but uses a different translation which expects all applications to be  $\beta$ -redexes. This allows  $\rightarrow_b$  reduction to be simulated easily. The upside to this approach is that PSN does not have to be reproven for the target calculus. The downside is that if the simulation is to be used for a proof of PSN for the source calculus then, having

established the simulation, we cannot immediately infer PSN as the translation is not the identity on pure terms in general. However, depending on the target calculus, inferring PSN may not be too difficult. For example, if  $t$  is a strongly normalising pure term then the bodies of substitution of  $\mathcal{T}(t)$  are strongly normalising and  $\downarrow_{gc}(\mathcal{T}(t)) = t$ . If the target calculus has PSN then this implies that  $t \in \text{SN}_{\lambda_2}$ . Bloo and Rose proved that these properties ensure that  $\mathcal{T}(t)$  is strongly normalising in  $\lambda_{xgc}$  [20]<sup>11</sup>. Our intuition is that if a pure term  $t$  is strongly normalising for  $\beta$ -reduction then  $\mathcal{T}(t)$  should be strongly normalising for any sensible target calculus with PSN.

To prove PSN using this simulation, the target calculus must have PSN and at least as strong a level of composition of substitutions as the source calculus. From our experience here, the second approach seems preferable if some type system exists to characterise the target calculus or if the pessimistic translation preserves strong normalisation (as in our argument for  $\lambda_{xgc}$  in the last paragraph). The first approach works well if the proof of PSN for the altered target calculus is simple.

This simulation is a technical tool; it would be far too inefficient for implementation due to the potentially exponential number of duplicate substitutions created through translation. However, we can use it to prove normalisation properties of the source calculus using properties of the target calculus. We presented two proofs of PSN in this chapter by applying the different approaches to  $\lambda_{lr}$  and  $\lambda_{es}$ . At the time of writing, no other calculi have PSN and FCS, both of which are required for a proof of PSN using the simulation.

We presented a characterisation of  $\text{SN}_{\lambda_{es}}$  and then used the simulation to prove that the same set characterised  $\text{SN}_{\Lambda_{sub}}$ , giving the first type-theoretic characterisation of  $\text{SN}_{\Lambda_{sub}}$ . This characterisation lends strong support to our claim that  $\Lambda_{sub}$  is remarkably close to the  $\lambda$ -calculus: from PSN we can conclude that all reducts of pure terms which are strongly normalising for  $\rightarrow_{\beta}$  are strongly normalising for  $\rightarrow_{bcgc}$ ; as  $\mathcal{C}(t) \twoheadrightarrow_b t$  and  $t \in \text{SN}_{\lambda_{es}}$  exactly when  $\mathcal{C}(t) \in \text{SN}_{\beta}$  (Lemma 9.48 and PSN), we can conclude that all terms which are strongly normalising for  $\rightarrow_{bcgc}$  are reducts of pure terms which are strongly normalising for  $\rightarrow_{\beta}$ . The same holds for  $\lambda_{es}$ .

## 9.6 Related work

There are many techniques for proving PSN for explicit substitution calculi. However, many of these take advantage of the inductive nature of local substitution *i.e.* that substitutions are propagated down into a term. Rose and Bloo's inductive methods [20, 133, 18] are tricky to

<sup>11</sup>We can not use  $\lambda_{xgc}$  as a target calculus for  $\Lambda_{sub}$  as the latter has FCS whilst the former does not. However, it could possibly be used as a target calculus in which to simulate  $\Lambda_{subc\odot}$  reduction as neither compose substitutions.

apply to a calculus with FCS. Likewise, we have not tried the recursive path ordering technique as we felt that complications may occur. Specifically, we felt that the fact that the substitution definition both persists after a  $\overline{C} \rightarrow$  reduction and also remains in place may be a complication. It is possible to prove PSN for  $\Lambda_{\text{sub}}$  directly by relating terms of  $\Lambda_{\text{sub}}$  terms with terms of  $\lambda_I$  and then applying Lengrand’s techniques [93]. Our first proof of PSN indirectly provides this method using our simulation in  $\lambda\text{blxr}$ . A proof using a direct relationship between  $\Lambda_{\text{sub}}$  and  $\lambda_I$  would be preferable and probably neater.

Intersection types [40, 41] characterise the strongly normalising terms of the  $\lambda$ -calculus [129]. Dougherty and Lescanne [53] used an intersection type system to characterise the set of terms which are normalising under leftmost reduction or head reduction in  $\lambda\text{xgc}$ . Lengrand et al. [94] advanced this by using an extended (without the universal type) type system to characterise the strongly normalising terms of  $\lambda\text{xgc}$ .

$\Lambda_{\text{sub}}$  has been encoded as a bigraphical reactive system (Brs) (see Section 11.1). Bundgaard and Hildebrandt [22] have encoded the *Higher-Order Mobile Embedded Resources (Homer)* calculus as a Brs with explicit substitution and garbage collection. They base their presentation on  $\Lambda\text{BIG}$  and prove an operational correspondence between Homer and their encoding. The natural approach to modelling substitution with explicit substitutions in bigraphical systems seems to be non-local substitution, employing the link graph structure to bind variables to their values rather than propagating explicit substitutions. Methods for reasoning about non-local substitution are still in infancy; we hope our simulation method is a useful contribution.

The connection between explicit substitution calculi with distributive rules for substitution and cut elimination in proof-nets has provided insights into both areas of research. It has yielded normalisation proofs for existing explicit substitution calculi [51, 52] as well as aided the development of new calculi with some or all of the properties of preservation of strong normalisation (PSN), open confluence, and full composition of substitutions [80, 78].

Kesner and Lengrand [79] have demonstrated the connection between higher-order substitution and proofs in linear logic by translating typed  $\lambda\text{lxr}$ -terms into proof-nets [61]. More recently, Kesner has demonstrated a similar relation between  $\lambda\text{es}$  and proof-nets. Given the strong similarities between  $\Lambda_{\text{sub}}$  and  $\lambda\text{es}$ <sup>12</sup>, the relationship between  $\Lambda_{\text{sub}}$  and proof-nets deserves attention. Kesner defines such a relationship by composing the translation of  $\Lambda_{\text{sub}}$  into  $\lambda\text{es}$  with the translation of  $\lambda\text{es}$  into proof nets and uses the respective simulations to define a simulation of  $\Lambda_{\text{sub}}$  reduction in proof nets. A similar relationship can be described using the simulation of  $\Lambda_{\text{sub}}$  reduction in  $\lambda\text{blxr}$ .

<sup>12</sup>PSN, FCS, open confluence, a simulation between them, and  $\text{SN}_{\Lambda_{\text{sub}}} = \text{SN}_{\lambda\text{es}}$ .

As these two relationships are defined using different methods of translation (modifying the target calculus versus the pessimistic translation), it would be worth investigating the differences between them. Our only caveat is that the translations into proof nets are suboptimal as they introduce extra substitutions and reduction steps.

## Summary of Part II

We have supported our hypothesis that  $\Lambda_{\text{sub}}$  has a close correspondence to the  $\lambda$ -calculus with proofs of (closed) confluence and preservation of strong normalisation. On a lesser note, we showed step-by-step simulation of  $\beta$ -reduction and full composition of substitution. Open confluence does hold but we omitted the proof here (we provided a citation). Terms typable in a simply typed or intersection typed discipline were shown to be terminating. Furthermore, the characterisation of  $\text{SN}_{\Lambda_{\text{sub}}}$  was given by extending the type system which characterises the strongly normalising pure terms. All in all this is quite remarkable for a calculus which was conceived primarily as a means to model the  $\lambda$ -calculus in bigraphs. To put this in perspective with the research in explicit substitution calculi with local substitution, only  $\lambda\text{es}$  (and possibly  $\lambda\text{lxr}$ ) shares all these properties.

Although we indirectly proved closed confluence of  $\Lambda\text{BIG}$ , this work does not add to a general theory of confluence for bigraph theory. However, the technical tools employed by Bloo and Rose in their proofs of closed confluence – projection and the generalised interpretation method – may be helpful in proving this property in other bigraphical explicit substitution encodings of confluent calculi.

Our main novel contribution in this section was the introduction of a new method of proving PSN by simulating non-local substitution with local substitution. Our first application of this technique involved altering the calculus with local substitution and reproving PSN in the new calculus. Our second application (joint work) avoids this extra proof of PSN but places more burden on the translation between the calculi.

## Part III

# Applications of Kind Sortings

---

## Outline of Part III

This part of the dissertation supports our final hypotheses.

We first show that the kind sorting of sites allows us to specify preconditions of parameters to parametric reaction rules, namely that we can guarantee the absence of exposed nodes of certain controls. We demonstrate this with examples of reaction rules with spatial preconditions. This reduction *in absentia* is exploited to show that a basic level of flow control can be built into the reaction relation. This notion is then applied to model simple bigraphical algorithms, from a flattened copy operation to an encoding of a Sudoku solver for simple puzzles.

The application of kind sorting to model semi-structured data is demonstrated by adding document order to an existing encoding of XML data using our interpretation of Milner’s notion of multi-nodes. We also sketch an encoding of BibTeX entries.

We then marry the two hitherto independent threads of the dissertation; the investigations of kind sorting and Milner’s explicit substitution calculus. We combine an interpretation of types as sets of controls with corresponding subcategories of kind sortings to model the untyped, simply typed, and intersection typed  $\lambda$ -calculus using the same sorting schema. We introduce a sorting to remove malformed bigraphs from Milner’s model of the  $\lambda$ -calculus which kind sorting does not remove. The results of Part II of the dissertation are then used to present confluence and normalisation properties of the bigraphical models with positive results.



# Chapter 10

## Expressiveness of Kind Sorting

*You're the night, Lilah*

*You're everything that we can't see*

*Lilah, you're the possibility*

*The Night – Morphine*

This chapter explores the expressiveness that kind sorting adds to bigraphs. We begin by demonstrating how the sorting of sites in kind parametric reaction rules can be used to guarantee the absence of exposed nodes of parameters to the rules *i.e.* kind sorting allows us to express simple placing preconditions for reaction. We then show how these preconditions can be used to introduce some flow control into the reaction relation by defining rules with mutually exclusive preconditions.

In Section 10.4, we consider the combination of kind sorting and a rigid control-sorting proposed by Birkedal et al. to model context-aware systems. We take their example of a location-aware printing system and present a kind signature which models the intended hierarchy as well as some new reaction rules which take advantage of the extra expressivity.

In Section 10.5, we alter their bigraphical tree-traversal ‘algorithm’ to define a new algorithm which presents a good case for what we call prioritised reaction firing. We also present a simple encoding of a distributed Sudoku reduction system which can solve some simple puzzles.

Finally, we demonstrate how the static structure of kind sortings may be used to model semi-structured data with an ordered tree structure by altering Conforti, Macedonio, and Sassone’s

encoding of XML data as well as describing a signature for representing BibTeX files.

## 10.1 Notation

In the examples of this chapter, our attention will usually be directed at the place graph. Therefore, we will omit some linking in the algebraic terms and the diagrams of reaction rules in order to present the examples as simply as possible.

We will also extend the term language of bigraphs to describe the set of controls that sites may contain. In the pure theory, a site is represented in a term by a square,  $\square$ , which may be indexed if necessary. For kind sorted bigraphs, sites in terms are denoted in one of three ways. Let  $\mathcal{K} = \{\mathbf{room}, \mathbf{prsn}, \mathbf{pc}\}$ . We will write  $\boxed{\mathbf{room} \mathbf{prsn}}$  to describe a site which can contain controls **room** and **prsn**. We may also write  $\boxed{\neg \mathbf{pc}}$ , or ‘not **pc**’ to describe the same site<sup>1</sup>. If a site can contain all elements of  $\mathcal{K}$ , we denote it simply by  $\square$ . To avoid any ambiguity, we will index sites when necessary.

## 10.2 In the absence of control...

This section describes the extra expressiveness that kind sorting allows reaction rules, concentrating mainly on the fundamental kind sorting. Kind reaction rules are more expressive than pure rules for one simple reason; by sorting the inner interface, we specify which controls can lie directly under the roots of parameters to parametric reaction rules in a kind Brs. This allows us to specify ground rules where the *absence* of exposed nodes of certain controls is guaranteed for the parameters; *in absentia*, the rule may fire, otherwise it cannot. The following examples highlight this notion although we do not claim that they are useful models in themselves. We omit details of signatures or link sortings which are unimportant to the examples.

### 10.2.1 Multiple readers

Consider the signature  $\mathcal{K} = \{\mathbf{F}, \mathbf{R}, \mathbf{W}\}$ , where the elements represent files, readers, and writers respectively. Let readers and writers be atomic controls and let files be able to contain readers and writers. The reaction rule of Figure 10.1 depicts a reader being granted access to file, represented by placing the reader inside the file. We can specify the inner sort of the rule as  $\{\mathbf{R}\}$ . Therefore, the rule states that a reader may gain access to a file if only readers (and not writers) have access

<sup>1</sup>In signatures with capacities, this means that  $kind(s)(\mathbf{pc}) = 0$  for the site  $s$ .

to the file. A signature with capacities would be able to set limits on the number of readers and writers which could access a file.

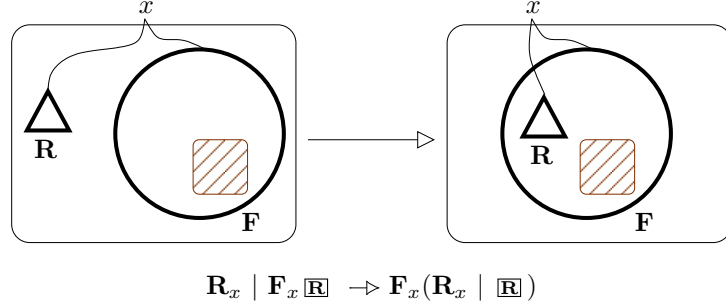


Figure 10.1: Multiple readers accessing a file

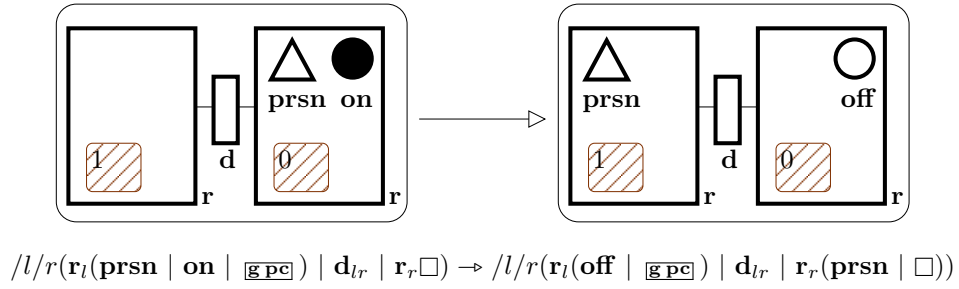


Figure 10.2: Lights that turn off when the last person exits the room

### 10.2.2 Smart buildings

This next rule was inspired by the DELCA example from the Bigraphical Programming Languages group page [66, 13].

Let  $\mathcal{K} = \{\mathbf{r}, \mathbf{d}, \mathbf{prsn}, \mathbf{g}, \mathbf{pc}, \mathbf{on}, \mathbf{off}\}$ , where the elements represent rooms, doors, people, Ghosts, computers, lit lights, and unlit lights respectively. The kind signature specifies that people, lights, and doors are atomic controls and that rooms are able to contain people, Ghosts, computers, and lights. Consider the reaction rule of Figure 10.2, representing a person moving from one room to another where the light in the first room turns off on exit. A pure Brs based on this rule describes a system where the lights switch off when a person leaves a room. Although energy-efficient, this could get incredibly annoying for the remaining people in the room. In a kind Brs, we can assign the sort  $\{\mathbf{g}, \mathbf{pc}\}$  to the topmost site; we purposefully omit the  $\mathbf{prsn}$  control. The rule now states

that if the *last* person in a room leaves (*i.e.* in the absence of any other person) then the light switches off – a behaviour perhaps more desirable.

We can model the correct behaviour using a pair of pure rules, one to account for the movement of people and another for switching off the lights in an empty room. There are two problems with this approach. First, we cannot allow sites in the room of the second rule (as parameters to these could contain people) so separate rules are required for each unique instance of a room and its contents. Second, as yet there is no notion of time in pure Brss so we cannot reason about when, or if, the second rule would fire.

### 10.2.3 Automatic teller machine

In this example, we consider the interaction between a user and an automatic teller machine (ATM). Let the controls **user**, **ATM**, and **vault** represent a user, an ATM, and an ATM's vault respectively and let the controls **rq50**, **50**, **20**, and **10** represent an internal ATM request for fifty euros, and a €50, €20, and €10 bank note. Let  $\mathcal{K}$  be a signature containing these controls with the containment structure of Figure 10.3(a). Imagine a scenario where a user has performed some interactions with the ATM. The ATM has an internal request for €50 to be made which is attached to some €50 note. The ATM then dispenses the note. This interaction is captured in Figure 10.3.

Figure 10.4 depicts two rules stating that if the machine has an internal request for €50 then this request is linked with some notes which total the requested sum. In a pure Brs, a user who requested €50 may receive either one €50 note or else two €20s and a €10 – there is a non-deterministic choice. If we instead sort the site of the second rule such that the sort does not contain **50**, the model specifies that we should dispense the larger notes first.

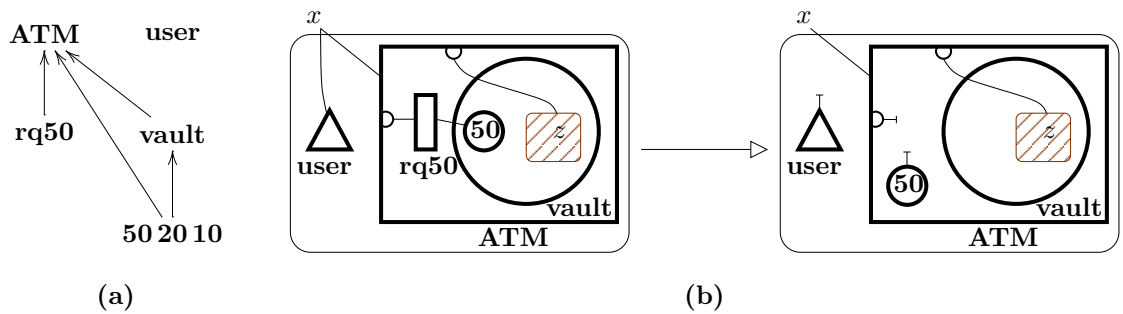


**Figure 10.3:** (a) The kind relation for the ATM example (b) The ATM returns €50

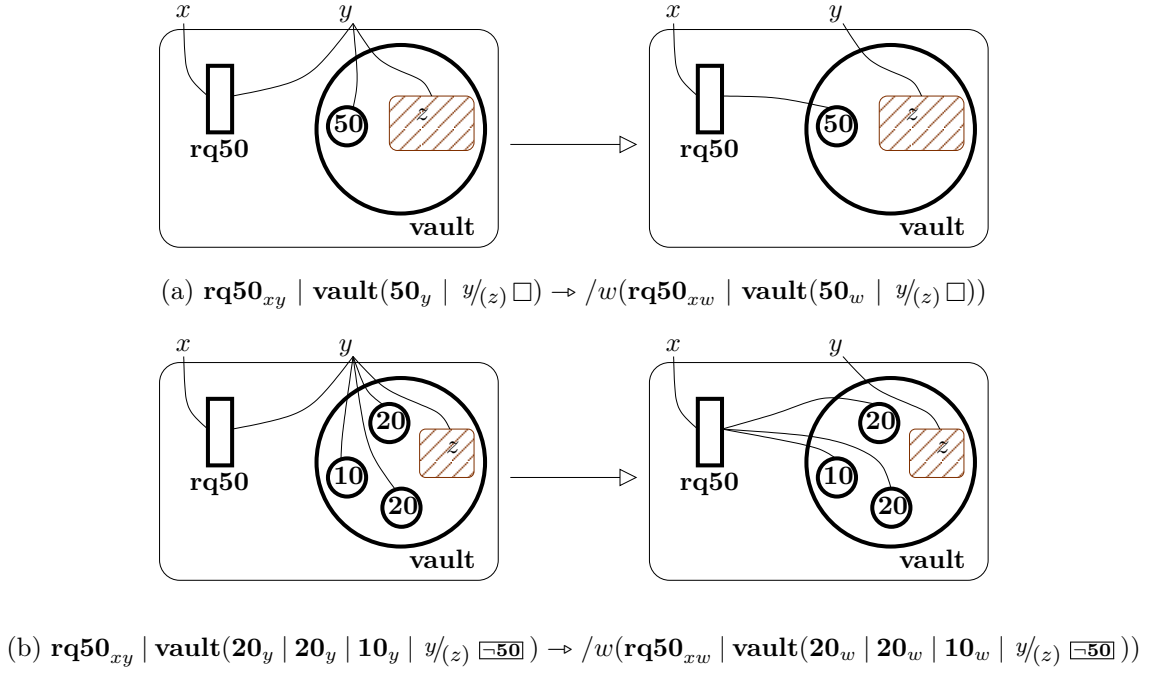


**Figure 10.4:** A request for €50 is linked with notes of that value

#### 10.2.4 Draughts

As well as modelling some formal grammars, kind sortings allow us to model simple games and their rules. The game of draughts can be modelled using a signature  $\mathcal{K} = \{\mathbf{B}, \mathbf{W}, \mathbf{KB}, \mathbf{KW}, \mathbf{S}, \mathbf{Trn}, \mathbf{Hop}\}$  where the elements represent black and white pieces and kings, squares, a ‘turn’ control, and a ‘hop’ control. Squares can contain pieces and kings and pieces and kings can contain turns and hops, as represented in Figure 10.5. Squares have arity four and all other controls have zero arity. We use the square tile sorting of Section 6.2.2 with four directions (NE, NW, SE, SW).

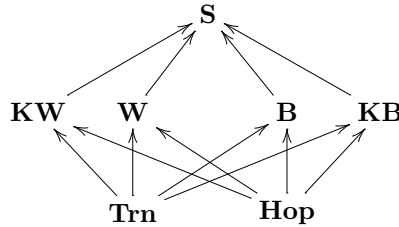


**Figure 10.5:** The kind relation for the draughts example

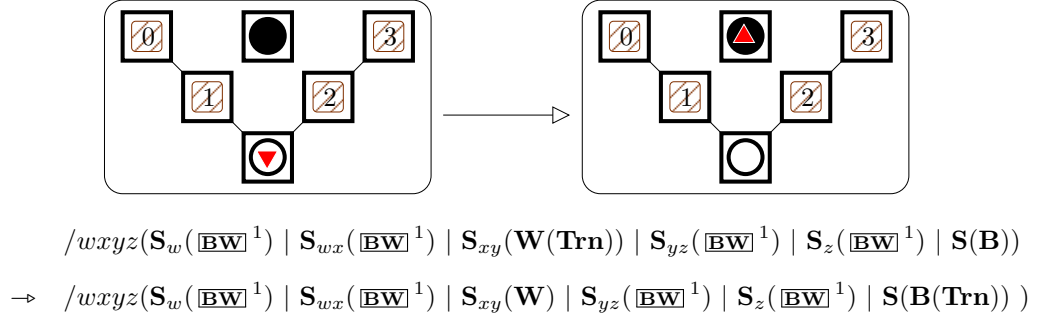


**Figure 10.6:** *A hop ends (all sites contain playing pieces)*

Some of the game rules are presented in Figures 10.6-10.10. To simplify the diagrams, we omit the linkages in some of the figures and (informally) let the position of places in the diagrams identify occurrences. In the remaining figures, it is assumed that any corner of a square not linked to another corner is uniquely linked to an outer name. A valid game of draughts is one in which the squares are linked in an  $n$  times  $n$  board pattern, where each square contains exactly one piece and there exists exactly one turn or hopper control (which by the signature is under some piece on the board)<sup>2</sup>.

In Figure 10.6, a player who was hopping has reached a point where there are no more hops available. The hop ends and the other player gets their turn. Figure 10.7 represents a black piece having reached the bottom of the board, whereupon the piece is crowned. Figure 10.8 represents a white piece moving forwards. A piece must contain a turn control, indicating that it is the player's turn to move. Upon moving forwards, the turn control is then transferred to a black piece. Figure 10.9 represents a player changing their mind as to which piece on the board they wish to move. Figure 10.10 represents a black player taking a white piece. Upon taking the piece, the turn control is changed to a hop control. This indicates that the player is in the middle of a series of hops.

Note that the game-board structure is imposed by linking rather than nesting. However, the contents of the squares are restricted by the kind sorting. Also, by using a sorting with semi-rigid maximum capacities, Figure 10.6 can specify that the sites contain at least one playing piece (we represent the maximum capacity of a site by subscripting the boxes in the term) Therefore, the rule assures that the player's hop ends if there are no more hops available.

<sup>2</sup>These last conditions could be enforced by using a kind sorting with semi-rigid maximum capacities.

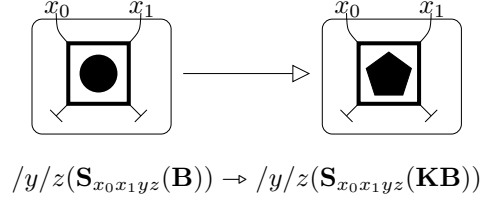


Figure 10.7: A piece is crowned

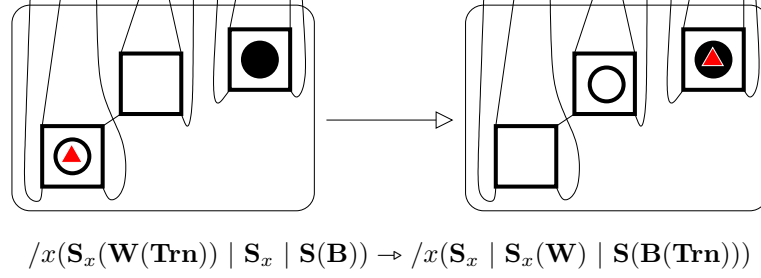


Figure 10.8: A piece advances

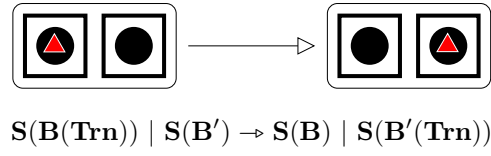


Figure 10.9: A player considers another move

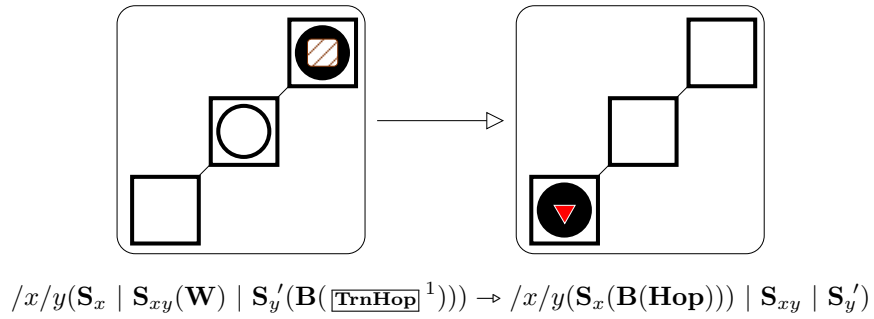


Figure 10.10: A player takes a piece

### 10.2.5 Boolean algebra

The signature  $\mathcal{K} = \{\mathbf{T}, \mathbf{F}, \wedge, \vee, \Rightarrow, \equiv, \neg\}$  represents a simple parallel boolean algebra computer where  $\wedge$  and  $\vee$ , respectively representing multiple conjunction and disjunction, are the only non-atomic controls and they can contain all controls in  $\mathcal{K}$  except their own control. Let  $ar(\mathbf{T}) = ar(\mathbf{F}) = ar(\wedge) = ar(\vee) = 1$ ,  $ar(\neg) = 2$ , and  $ar(\Rightarrow) = ar(\equiv) = 3$ . Call the first port of any control its ‘tag-port’<sup>3</sup> and any other port a ‘link-port’. We assume a link-sorting over a tag and a link sort where a link contains at most one tag and one link.

Figures 10.11-10.15 show some reaction rules representing the reduction of boolean terms to either true or false over this signature. We do not show all the reaction rules for implication, equivalence or negation but the missing rules are similar to those presented. The encoding is trivial and we will not dwell on it except to make two remarks.

The first remark is that the conjunction ( $\wedge$ ) and disjunction ( $\vee$ ) operators are represented by non-atomic controls in contrast with the other binary operators. If we represented them as atomic controls then we may as well use a link reactive system and drop the place graphs. However, we represent conjunction and disjunction as non-atomic controls for good reason; they are associative and commutative operators (unlike implication) and their evaluation depends on either all sub-evaluations having one value or else some sub-evaluation having the other *e.g.* the conjunction of  $n$  boolean values is false if at least one value is false and is true otherwise. This is exploited in rules 10.11 and 10.12 to allow efficient reaction rules where a conjunction of  $n$  variables may reduce to false as soon as any of its conjuncts does so.

The second remark does not concern kind sorting. The first reaction rule for implication covers both cases where the precedent is false. Any term that is connected to the ‘antecedent port’ of the implication via composition has its link severed after reaction – its tag port is then connected to an edge and it is essentially a useless term. The garbage collection ‘rule’ is a schema for a rule that can be described as “when a term is closed off, remove the term”<sup>4</sup>.

<sup>3</sup>This terminology is taken from the encoding of the  $\lambda$ -calculus in bigraphs [107].

<sup>4</sup>We realised (post-submission) that this model can be simplified using kind sorting with rigid capacities and multi-nodes. Negation can be modelled as a non-atomic node with capacity one. Conjunction, disjunction, and equivalence can be modelled as non-atomic nodes with capacity  $\odot$ . This better reflects the symmetry of equivalence (the model above orders the sides of an equivalence). If we then model implication as a 2-node control with inner capacities of one where the first chamber contains the antecedent and the second chamber contains the consequent, we can remove all links from the model. This is more intuitive; classical propositions have no notion of linkage. Furthermore, the first implication rule does not leave any ‘garbage’ and so we do not require the garbage collection rule. Note that we have modelled the commutative, associative operators using unordered controls and the non-commutative operator using an ordered multi-node control.



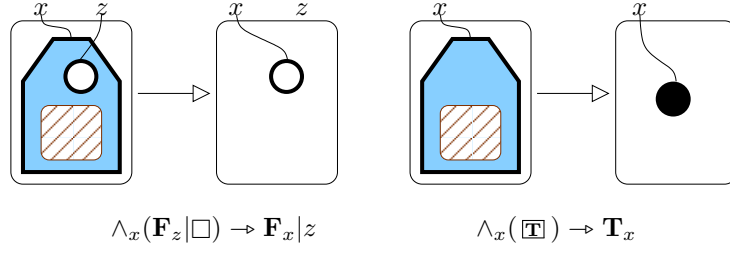


Figure 10.11: Boolean AND

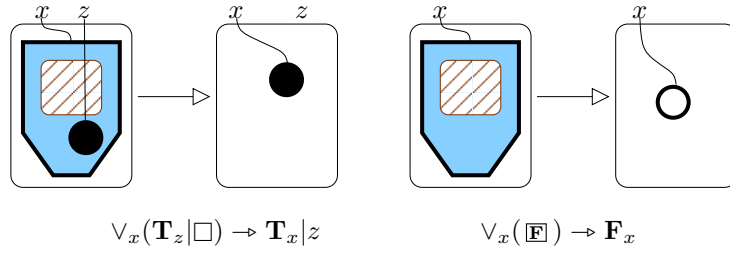


Figure 10.12: Boolean OR

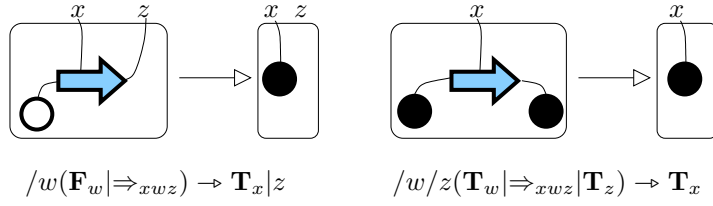


Figure 10.13: Boolean IMPLIES

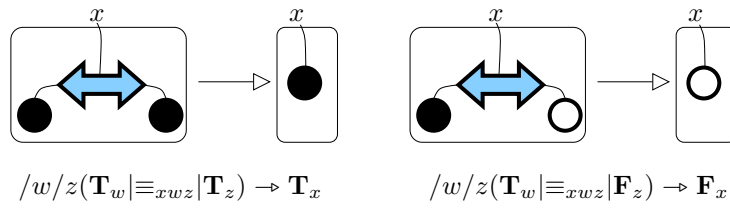


Figure 10.14: Boolean EQUIV

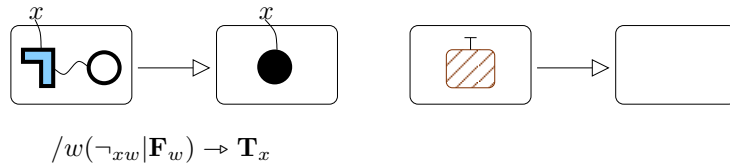


Figure 10.15: Boolean NOT and a garbage collection 'rule'

### 10.2.6 Ain't Got No/I Got Tea

This last example should not be taken seriously! In computer adventure games, players must navigate through a map, collecting and dropping items on their quest. We represent the basic structure of such a game using tiled link sorting and kind sorting.

Consider a signature  $\mathcal{K}$  consisting of a room control **room** with arity four and the obvious tiled link sorting,  $n$  controls **items** = {**item1**, **item2**, ..., **itemn**} of arity one representing items, a control **ad** of arity one representing players, controls **cs** and **notea** representing common sense and 'no tea', and a control **move** of arity two representing the actions take/drop with the containment relation given in Figure 10.16 where **items** represents all items *i.e.* the graph states that a room or a player may contain any item.

The two rules in Figure 10.17 represent the symmetric rules 'take' and 'drop' which move items in and out of a player's inventory (some linkage is omitted). Let **tea** be an item. Figure 10.18 represents the situation "if a player has no common sense and no tea then they may drop no tea, which becomes tangible." Thus, we may pick up, and simultaneously have, both tea and no tea.<sup>5</sup>

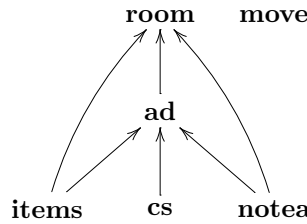
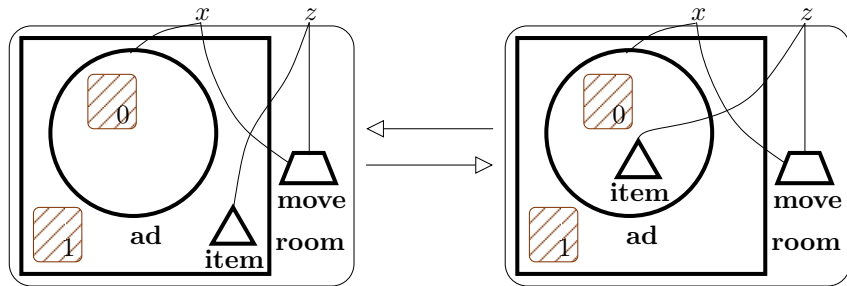


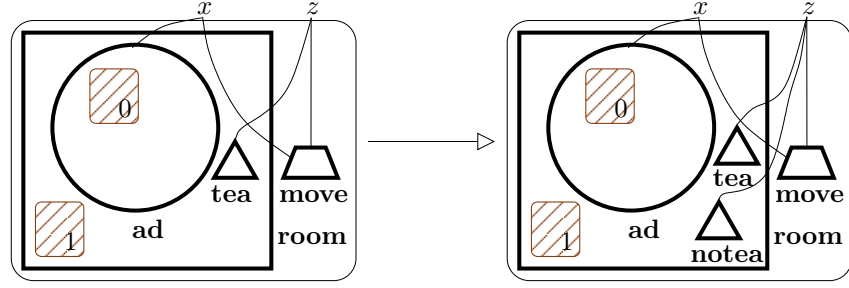
Figure 10.16: The kind relation for the adventure game example



$$\text{room}(\text{ad}_x \square_0 | \text{item}_z | \square_1) | \text{move}_{zx} \leftrightarrow \text{room}(\text{ad}_x (\text{item}_z | \square_0) | \square_1) | \text{move}_{zx}$$

Figure 10.17: Taking and dropping items

<sup>5</sup> *The Hitch Hiker's Guide to the Galaxy*, Infocom, 1984.



$$\text{room}(\text{ad}_x(\neg\text{tea}\neg\text{cs}))|\text{tea}_z|\square|\text{move}_{zx} \rightarrow \text{room}(\text{ad}_x(\neg\text{tea}\neg\text{cs}))|\text{tea}_z|\text{notea}_z|\square|\text{move}_{zx}$$

Figure 10.18: Drop ‘no tea’

### 10.3 ...be determined

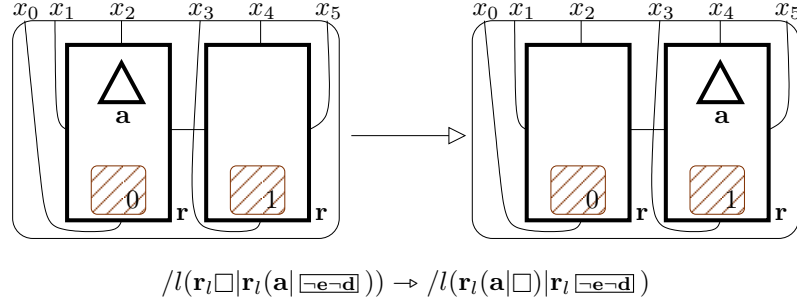
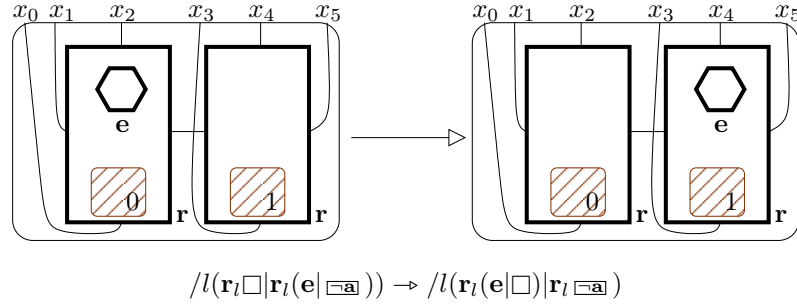
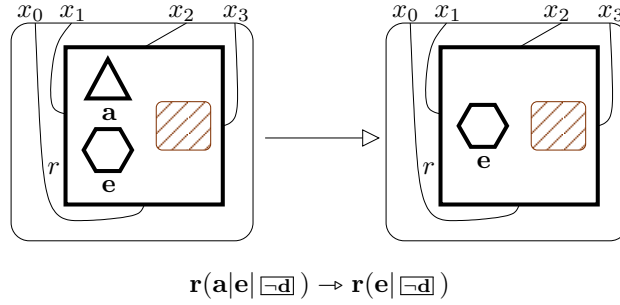
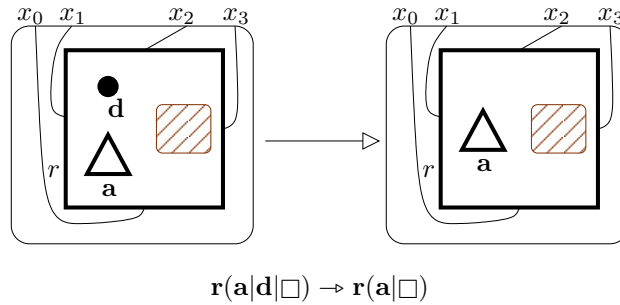
We now return to one of our examples in Chapter 3 to explain how determinism may sometimes be encoded into the reduction rules; we call this notion *prioritised reaction firing*. We will describe the rules of a simple tile-based game like Pac-Man<sup>®</sup> which consists of a series of square rooms containing dots and mobile agents and enemies. We assume without further comment that the rooms are linked together using a tiled link sorting. Agents move around the rooms collecting dots and enemies move around killing agents. The rules of the game are as follows: agents cannot leave a room containing a dot without consuming it and cannot leave a room containing an enemy; enemies cannot leave a room with agents in it and cannot kill an agent before the agent consumes a nearby dot. We will show how kind sorting allows us to capture these rules.

The signature  $\mathcal{K}$  consists of a set of controls  $\{\mathbf{r}, \mathbf{a}, \mathbf{e}, \mathbf{d}\}$ , representing rooms, agents, enemies, and dots respectively. The containment hierarchy is captured by the mapping

$$\mathbf{r} \mapsto \{\mathbf{a}, \mathbf{e}, \mathbf{d}\}, \quad \mathbf{a} \mapsto \emptyset, \quad \mathbf{e} \mapsto \emptyset, \quad \mathbf{d} \mapsto \emptyset.$$

The rules for the game are encoded as reaction rules. The ‘move north’ rules are shown in Figures 10.19 and 10.20. For the agent movement rule, we have specified that the room in which the agent starts has no dots or enemies. For the enemy movement rule, we stated that the starting room has no agents. Figure 10.21 depicts the rule where an enemy kills an agent in the same room where the room does not contain any dots and Figure 10.22 shows the rule where an agent collects a nearby dot. In these rules, the sites state which controls are guaranteed to be absent. The constraints of the game are thus modelled by the *guaranteed absence* of certain entities.

An interesting aspect of this reactive system is that the dynamic behaviour of an agent can be described as:

**Figure 10.19:** An agent moves between two rooms**Figure 10.20:** An enemy moves between two rooms**Figure 10.21:** An enemy eliminates an agent**Figure 10.22:** An agent collects a dot

| <b>a</b> | <b>d</b> | <b>e</b> | <b>a ∧ d</b> (collect) | <b>a ∧ ¬d ∧ e</b> (die) | <b>a ∧ ¬d ∧ ¬e</b> (move) |
|----------|----------|----------|------------------------|-------------------------|---------------------------|
| T        | T        | T        | T                      |                         |                           |
| T        | T        | F        | T                      |                         |                           |
| T        | F        | T        |                        | T                       |                           |
| T        | F        | F        |                        |                         | T                         |

**Table 10.1:** Truth table representing mutually exclusive reaction rules (empty rows omitted)

```

if dot in room, collect dot
else if ghost in room, die
else move.

```

In short, the restrictions we have placed on what the sites can contain have prioritised the firing of reactive rules and allowed some basic flow control. The reason we get this clean separation (if, else if, else) can be easily explained when we consider the extra information sorting the sites gives us in terms of propositions of a spatial logic.

Let the proposition **a** denote ‘there is an agent here’ and  $\neg\mathbf{a}$  denote ‘there are no agents here’. Define propositions **d** and **e** and their negations similarly. The firing conditions for the rules for an agent moving, dying, or consuming dots can then be represented as conjunctions of these propositions from the point of view of an agent. For example, in the redex of the ‘collect’ rule, there is definitely one dot and one agent in the agent’s room. The firing condition is then  $\mathbf{a} \wedge \mathbf{d}$ . In the redex of the ‘move’ rule, there is definitely one agent and no dots or enemies in the agent’s room. The firing condition is then  $\mathbf{a} \wedge \neg\mathbf{d} \wedge \neg\mathbf{e}$ . The ‘collect’, ‘die’, and ‘move’ rules have been designed so that their firing conditions are mutually exclusive. This is shown in Table 10.1. The conjunction of the three firing conditions is false; there is no overlap between the rules.

Therefore, kind bigraphs can allow some basic flow control to some systems based on logical propositions on the spatial structure of redexes.

## 10.4 Context-aware systems

Birkedal et al. have proposed a method for modelling context-aware systems in bigraphs by separating perceived and actual content by means of a shared proxy. This method is achieved using a *Plato-graphical sorting*, a special case of their safe rigid control-sortings we discussed in Section 6.3.2. In that section, we showed that kind sorting can be safely combined with rigid

control-sortings. In this section, we consider the combination of kind sorting with Plato-graphical sorting which we believe useful due to the expressivity of reaction rules.

As a motivating example, we consider Birkedal et al.'s Location-aware Printing System [15] which extends Braione and Picco's model [21] with location-awareness. Our kind function for the signature is depicted in Figure 10.23. The system is split into three overlapping subsystems.

The *context*  $C$  consists of a hierarchy of physical locations **loc** which may contain printers **prt** (which themselves may contain some binary data **dat**) and devices **dev** representing *e.g.* PDAs or PCs. Two types of printers are modelled, distinguished by whether a **pcl** or **raw** node lies inside a **prt** node. The *context*  $P$  models a pool of pending print jobs **jobs** (containing documents **doc**), printers, and **pcl** and **raw** printer tables **prts** which keep track of live printers in the system. The *agent*  $A$  can create new print jobs. Note that the context is only concerned with the physical locations of printers and devices, the proxy is concerned with printers and print jobs but not physical locations, and the agent is only concerned with the print jobs and does not care about the number or locations of printers. We will not repeat all the reaction rules of the system here but they model (dis)appearance of printers, dispatch of print jobs to printers, completion of print jobs, discovery of new printers, and creation of new print jobs.

Our first application of kind sorting is to correctly model the hierarchy and capacities of the system with semi-rigid maximum capacities, depicted in the figure. We also take advantage of our expression of absence. Consider the original reaction rule of the Location-aware Printing System<sup>6</sup>

$$\begin{aligned} &\mathbf{jobs}(\mathbf{doc}_{z,x}|-_0)\|\mathbf{prts}_y(\mathbf{pcl})\|\mathbf{loc}(\mathbf{dev}_x|\mathbf{prt}_y(\mathbf{pcl})) \longrightarrow \\ &\mathbf{jobs}(-_0)\|\mathbf{prts}_y(\mathbf{pcl})\|\mathbf{loc}(\mathbf{dev}_x|\mathbf{prt}_y(\mathbf{pcl}|\mathbf{dat}_z)). \end{aligned} \quad (5')$$

where a print job ( $\mathbf{doc}_{z,x}$ ) in a pool of pending jobs (**jobs**) is not sent to a printer ( $\mathbf{prt}_y$ ) unless the printer and the device which submitted the job ( $\mathbf{dev}_x$ ) are colocated. In the reactum, the print job has been sent to the printer as  $\mathbf{dat}_z$ . The first change we will make is trivial; we will just add a site into the rule. This site allows us to consider two new rules below.

$$\begin{aligned} &\mathbf{jobs}(\mathbf{doc}_{z,x}|-_0)\|\mathbf{prts}_y(\mathbf{pcl})\|\mathbf{loc}(\mathbf{dev}_x|\mathbf{prt}_y(\mathbf{pcl})|-_1) \longrightarrow \\ &\mathbf{jobs}(-_0)\|\mathbf{prts}_y(\mathbf{pcl})\|\mathbf{loc}(\mathbf{dev}_x|\mathbf{prt}_y(\mathbf{pcl}|\mathbf{dat}_z)|-_1). \end{aligned} \quad (5'_2)$$

Rules (5') and (5'<sub>2</sub>) above can capture the behaviour where a document belonging/linked to a user (modelled as a **dev** node) is printed when the user enters a location containing a printer. This raises the question of what to do with printers when there are no users nearby. Perhaps we would like to switch them off when there are no users present and wake them up when one arrives.

<sup>6</sup>We use the alternative term notation  $-_i$ ,  $i \in m$  rather than  $\square_i$  to represent sites here.

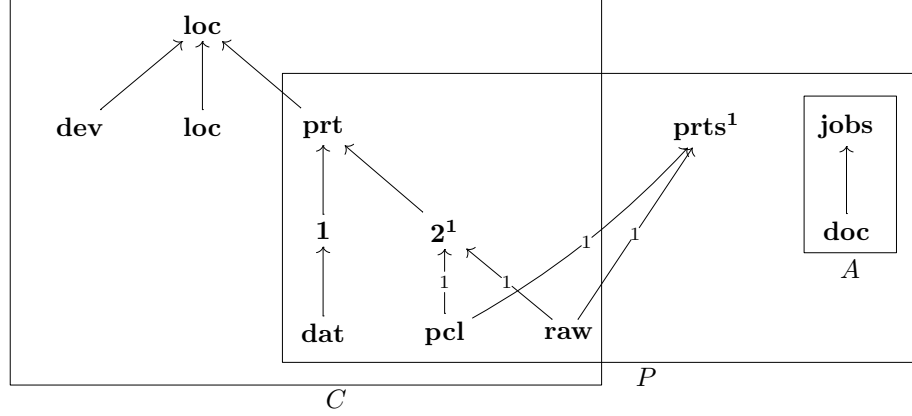


**Figure 10.23:** Kind relation for the Location-aware Printing System

Assume a new control  $\mathbf{prt}^{\text{off}}$  with the same kind sort as  $\mathbf{prt}$  which models a printer in power-saving mode. The parametric rule for putting printers to sleep (one by one) can be written as:

$$\mathbf{loc}(\mathbf{prt}_y(-_0) | \boxed{\neg \mathbf{dev}}_1) \longrightarrow \mathbf{loc}(\mathbf{prt}_y^{\text{off}}(-_0) | \boxed{\neg \mathbf{dev}}_1).$$

The rule for waking the printers up (again, one by one) is:

$$\mathbf{loc}(\mathbf{prt}_y^{\text{off}}(-_0) | \mathbf{dev}_x|-_1) \longrightarrow \mathbf{loc}(\mathbf{prt}_y(-_0) | \mathbf{dev}_x|-_1).$$

These rules take advantage of the spatial propositions “no users here” and “a user is here”, the first supplied by the kind sorting.

These two rules partially answers the problem mentioned by Birkedal et al., that “*it is generally very difficult... to observe the absence of something in the context directly.*” Kind sorting allows us to guarantee the absence of exposed nodes of certain controls in parameters to reaction rules.

## 10.5 Algorithms

In this section, we apply the notion of reaction *in absentia* and the resulting flow control to model some simple bigraphical algorithms.

### 10.5.1 “Find them and kill them”

Consider a hierarchy of locations which contain devices where we want to ask “which devices are present within location  $l$ .” There is no simple, internal way to answer this question using bigraph

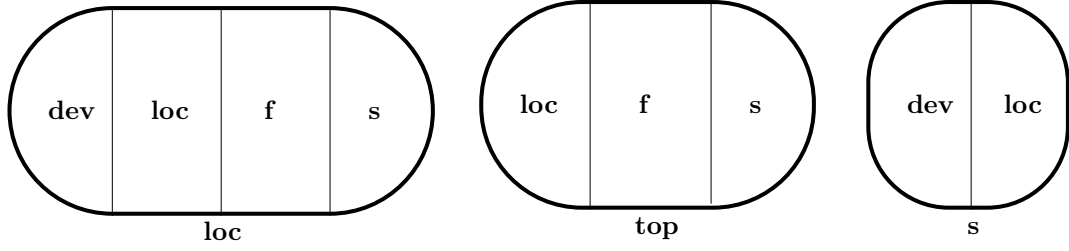


**Figure 10.24:** Multi-nodes for the “copy all devices” query

theory. However, Birkedal et al. have presented a set of bigraphical rules which combine as a bigraphical tree-traversal ‘algorithm’ to implement such a query [14]. The algorithm works by using a pair of controls **f** and **s** to traverse a tree of locations which find, one-by-one, devices and add them to the output of the query.

We can use their technique to define a similar algorithm to implement an operation which finds and moves (or copies or deletes) all devices in a building to an output location. In the context of their model, this set of rules does not make much sense but it could be interpreted as a flattening algorithm, taking a tree-like data structure and returning an unordered set of its elements. It demonstrates how certain operations can be done in one step using a kind sorting with multi-nodes. Further, their example was based on the assumption that locations can only contain either devices or other locations – this assumption can be enforced using a kind sorting.

The kind relation for the signature is shown in Figure 10.25 with locations **loc** and devices **dev**. The multi-node **top** (defined like **loc** except that it is missing the **dev** section) represents a top location. The controls **in** and **out** represent input and output nodes. The control **g** is a dummy control used to represent that a query operation is in progress. The control **f** is used to mark the location where the algorithm is currently operating. The control **s** encodes a tree-like structure, used to keep track of the locations in the tree which have been visited by collecting them.

We will use a kind sorting with semi-rigid maximum capacities. The combination of multi-nodes with kind sorting allows us to separate **loc** and **s**-nodes into areas which can each contain one type of control. We consider multi-nodes as ‘capsules’ having multiple sites, ordered in the obvious way  $(\mathbb{N}, \leq)$  and write them as *e.g.*  $\mathbf{loc}_x(-_0 \parallel -_1 \parallel -_2 \parallel -_3)$ . These capsules are depicted in Figure 10.24. When a section of a multi-node is empty, we denote this with a blank space so that  $\mathbf{loc}_x(\parallel \parallel \parallel)$  denotes a multi-node **loc** with no contents.

The set of rules for the algorithm is given in Figure 10.26. The algorithm implements a copy operation where the hierarchy is not copied (a ‘flattened copy’). Instead of taking copying one



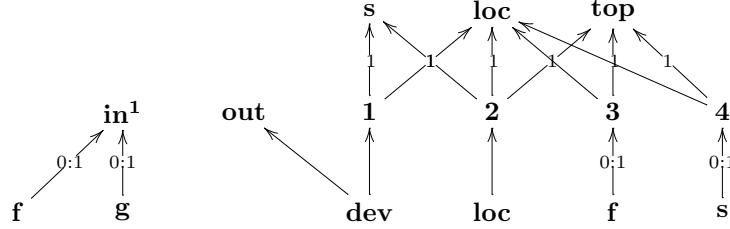


**Figure 10.25:** Kind relation for the “copy all devices” query

device at a time as would be done in pure bigraphs, the fact that a site only contains **dev** controls means that we can be more efficient and copy the entire site to the output node. Using a subcategory where no capacities are zero, we avoid copying empty sites (which would otherwise create infinite loops) by requiring that the site contains a minimum of one control.

Table 10.2 describes the criteria for firing the rules from the point of view of the **f** node. The table demonstrates that the rules fire under separate criteria with some flow control which is why we call the set of rules an algorithm. However, the algorithm has some non-determinism as the digging rules allow any colocated location to be explored.

The sequence in Figure 10.27 shows this algorithm at work on the tree

$$InitialTree = \mathbf{top}(\mathbf{loc}(\mathbf{dev}_1 \parallel \mathbf{loc}(\mathbf{dev}_2) \mid \mathbf{loc}(\mathbf{dev}_3 \mid \mathbf{dev}_4)) \mid \mathbf{loc}()).$$

We unambiguously omit the names and empty sections of multi-nodes to make the example easier to read and colour the node currently holding **f** in blue. We write **dev** as **d** to save space. When following the sequence, it may help to recognise that according to the rules, an **f**-node can only dig or climb if there are no colocated devices and can only climb to the top if there are no colocated devices or locations. This expressiveness in the rules comes from the use of the specific multi-nodes **loc**, **top**, and **s** where each section of the multi-node capsule contains a unique type of control.

One assumption we have made in the rules of the algorithm is that there is one *f* query propagating through the hierarchy. The rules could be modified to allow simultaneous queries but these may interfere with each other. One possible solution is to add capacities to rigid control-sorting which would allow us to specify that at most one *f* query can exist below a given root and to use a link sorting where inputs and outputs are linked to at most one query.

initialise

$$\mathbf{in}(\mathbf{f}) \parallel \mathbf{top}(-_0 \parallel \parallel) \parallel \mathbf{out}()$$

$$\longrightarrow \mathbf{in}(\mathbf{g}) \parallel \mathbf{top}(-_0 \parallel \mathbf{f} \parallel \mathbf{s}(\parallel)) \parallel \mathbf{out}()$$

dig from top

$$\mathbf{top}(\mathbf{loc}_x(-_1 \parallel -_2 \parallel \parallel) \mid -_0 \parallel \mathbf{f} \parallel \mathbf{s}(\parallel -_3))$$

$$\longrightarrow \mathbf{top}(\mathbf{loc}_x(-_1 \parallel -_2 \parallel \mathbf{f} \parallel \mathbf{s}(\parallel)) \mid -_0 \parallel \parallel \mathbf{s}(\parallel -_3))$$

copy colocated devices

$$\mathbf{loc}_x(-_0 \parallel -_1 \parallel \mathbf{f} \parallel \mathbf{s}(\parallel -_2)) \parallel \mathbf{out}(-_3)$$

$$\longrightarrow \mathbf{loc}_x(\parallel -_1 \parallel \mathbf{f} \parallel \mathbf{s}(-_0 \parallel -_2)) \parallel \mathbf{out}(-_3 \mid -_0)$$

dig

$$\mathbf{loc}_x(\parallel \mathbf{loc}_y(-_3 \parallel -_4 \parallel \parallel) \mid -_0 \parallel \mathbf{f} \parallel \mathbf{s}(-_1 \parallel -_2))$$

$$\longrightarrow \mathbf{loc}_x(\parallel \mathbf{loc}_y(-_3 \parallel -_4 \parallel \mathbf{f} \parallel \mathbf{s}(\parallel)) \mid -_0 \parallel \parallel \mathbf{s}(-_1 \parallel -_2))$$

climb

$$\mathbf{loc}_x(\parallel \mathbf{loc}_y(\parallel \parallel \mathbf{f} \parallel \mathbf{s}(-_0 \parallel -_1)) \mid -_2 \parallel \parallel \mathbf{s}(-_3 \parallel -_4))$$

$$\longrightarrow \mathbf{loc}_x(\parallel -_2 \parallel \mathbf{f} \parallel \mathbf{s}(-_3 \parallel \mathbf{loc}_y(-_0 \parallel -_1 \parallel \parallel) \mid -_4))$$

climb to top

$$\mathbf{top}(\mathbf{loc}_x(\parallel \parallel \mathbf{f} \parallel \mathbf{s}(-_0 \parallel -_1)) \mid -_2 \parallel \parallel \mathbf{s}(\parallel -_3))$$

$$\longrightarrow \mathbf{top}(-_2 \parallel \mathbf{f} \parallel \mathbf{s}(\parallel \mathbf{loc}_x(-_0 \parallel -_1 \parallel \parallel) \mid -_3))$$

clean up

$$\mathbf{in}(\mathbf{g}) \parallel \mathbf{top}(\parallel \mathbf{f} \parallel \mathbf{s}(\parallel -_0))$$

$$\longrightarrow \mathbf{in}() \parallel \mathbf{top}(-_0 \parallel \parallel)$$

Figure 10.26: The set of rules for the copy algorithm

| Rule                   | Do when                                                        |
|------------------------|----------------------------------------------------------------|
| initialise             | query is pending                                               |
| dig from top           | at top, locations here                                         |
| copy colocated devices | not at top, devices here                                       |
| dig                    | not at top, no devices here, locations here                    |
| climb                  | not at top, no devices here, no locations here, location above |
| climb to top           | below top, no devices here, no locations here                  |
| clean up               | at top, no locations here                                      |

Table 10.2: The behaviour of the algorithm from the point of view of  $\mathbf{f}$ 

$$\begin{aligned}
& \mathbf{in}(\mathbf{f}) \parallel \text{InitialTree} \parallel \mathbf{out}() \\
&= \mathbf{in}(\mathbf{f}) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4)) \parallel \mathbf{loc}()) \parallel \mathbf{out}() \\
\rightarrow \text{initialise} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4)) \parallel \mathbf{loc}()) \parallel \mathbf{f} \parallel \mathbf{s}()) \parallel \mathbf{out}() \\
\rightarrow \text{dig from top} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4)) \parallel \mathbf{loc}(\mathbf{f} \parallel \mathbf{s}()) \parallel \mathbf{s}()) \parallel \mathbf{out}() \\
\rightarrow \text{climb to top} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4)) \parallel \mathbf{f} \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}() \\
\rightarrow \text{dig from top} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4) \parallel \mathbf{f} \parallel \mathbf{s}()) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}() \\
\rightarrow \text{copy colocated} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{d}_2) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4) \parallel \mathbf{f} \parallel \mathbf{s}(\mathbf{d}_1)) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d}_1) \\
\rightarrow \text{dig} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{d}_2 \parallel \mathbf{f} \parallel \mathbf{s}()) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4) \parallel \mathbf{s}(\mathbf{d}_1)) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d}_1) \\
\rightarrow \text{copy colocated} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{f} \parallel \mathbf{s}(\mathbf{d}_2)) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4) \parallel \mathbf{s}(\mathbf{d}_1)) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d}_1 \mid \mathbf{d}_2) \\
\rightarrow \text{climb} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4) \parallel \mathbf{f} \parallel \mathbf{s}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2))) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d}_1 \mid \mathbf{d}_2) \\
\rightarrow \text{dig} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4 \parallel \mathbf{f} \parallel \mathbf{s}()) \parallel \mathbf{s}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2))) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d}_1 \mid \mathbf{d}_2) \\
\rightarrow \text{copy} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{loc}(\mathbf{f} \parallel \mathbf{s}(\mathbf{d}_3 \mid \mathbf{d}_4)) \parallel \mathbf{s}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2))) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d}_1 \mid \mathbf{d}_2 \mid \mathbf{d}_3 \mid \mathbf{d}_4) \\
\rightarrow \text{climb} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{loc}(\mathbf{f} \parallel \mathbf{s}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4))) \parallel \mathbf{s}(\mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d}_1 \mid \mathbf{d}_2 \mid \mathbf{d}_3 \mid \mathbf{d}_4) \\
\rightarrow \text{climb to top} \quad & \mathbf{in}(g) \parallel \mathbf{top}(\mathbf{f} \parallel \mathbf{s}(\mathbf{loc}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4)) \parallel \mathbf{loc}())) \parallel \mathbf{out}(\mathbf{d}_1 \mid \mathbf{d}_2 \mid \mathbf{d}_3 \mid \mathbf{d}_4) \\
\rightarrow \text{clean up} \quad & \mathbf{in}() \parallel \mathbf{top}(\mathbf{loc}(\mathbf{d}_1 \parallel \mathbf{loc}(\mathbf{d}_2) \parallel \mathbf{loc}(\mathbf{d}_3 \mid \mathbf{d}_4)) \parallel \mathbf{loc}()) \parallel \mathbf{out}(\mathbf{d}_1 \mid \mathbf{d}_2 \mid \mathbf{d}_3 \mid \mathbf{d}_4) \\
&= \mathbf{in}() \parallel \text{InitialTree} \parallel \mathbf{out}(\mathbf{d}_1 \mid \mathbf{d}_2 \mid \mathbf{d}_3 \mid \mathbf{d}_4)
\end{aligned}$$

Figure 10.27: Example run of the flattened copy operation

### 10.5.2 Sudoku

Sudoku is a popular grid-based puzzle. The player is presented with a grid, typically a  $9 \times 9$  grid, with some squares filled with the digits 1 through 9. Besides rows and columns, the grid is further divided into nine  $3 \times 3$  blocks from the top-left of the grid to the bottom-right. An example puzzle is displayed in Figure C.1. The goal is to fill the empty squares with digits so that each row, column, and block each contain the digits 1 through 9. The solution to the example is given in Figure C.6<sup>7</sup>.

The player solves a Sudoku puzzle using logical reasoning. The puzzle comes in a variety of different difficulty levels. The simpler games can be solved using the reasoning in the algorithm below. The more difficult games require that the player test alternate possibilities to find the solution. A proper Sudoku game has a unique solution and we assume here that we are working with such games.

An algorithm to solve simple Sudoku puzzles is given below. We begin by marking the empty squares in the initial puzzle with possible values. When the content of a square is known, we call it a *filled* square. Otherwise, we call it a *marked* square.

**The algorithm:**

1. Mark each empty square with the numbers 1 through 9.
2. If a filled square contains a number  $n$ , remove the mark  $n$  from each marked square in the same row, column, and block as the filled square.
3. If only one marked square contains the mark  $n$  in a row (resp. column, block), the square becomes a filled square containing  $n$ .
4. Repeat steps 2 and 3 until none are possible.

This is a simple approach to the puzzle. We can use kind sorting to describe a Sudoku bigraphical reactive system which effectively encodes this algorithm. Deterministic algorithms exist for such a purpose [146] but the bigraphical approach immediately yields a non-deterministic, distributed algorithm.

#### Encoding Sudoku in bigraphs

The encoding simplifies previous work [118] based on suggestions from Mikkel Bundgaard.

---

<sup>7</sup>Each solution is also a latin square.

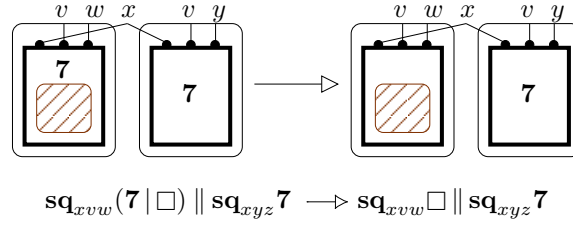
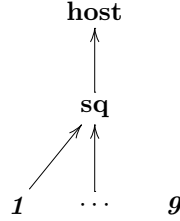


Figure 10.28: Removing possibilities

**Definition 10.1.** The kind  $\mathbf{Brs} \ \mathbf{BIG}(\Sigma, \mathcal{R}_S)$  has kind signature  $K_S$  consisting of nine atomic controls,  $1, 2, \dots, 9$  of arity zero, a passive non-atomic control  $\mathbf{sq}$  of arity three, and an active non-atomic control  $\mathbf{host}$  of arity zero. The kind function is described by the following graph:



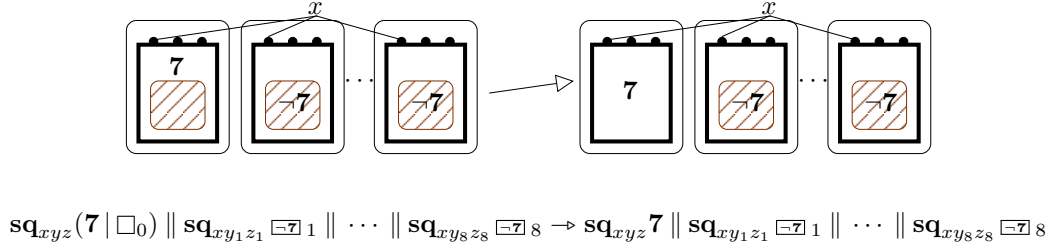
The rule set  $\mathcal{R}_S$  consists of:

- the twenty-seven rules described by Figure 10.28 with the identity instantiation  $\rho : 1 \rightarrow 1$
- the twenty-seven rules described by Figure 10.29 with the instantiation  $\rho : 8 \rightarrow 9$  defined as taking  $n \mapsto n$ ,  $n \in \{0, 1, \dots, 7\}$ .

We represent possible values of Sudoku squares with the digit controls. The **host** control will be discussed later. The squares themselves are represented with a **sq** control of arity three. The three ports of a square represent connections to rows, columns, and blocks. If the row ports on two squares are connected to the same name, we think of them as being in the same row. We will informally assume a plain link sorting with three sorts for rows, column, and blocks so that row ports may only link to each other *etc.* This sorting still allows bigraphs which do not conform to the game; a better sorting would enforce a maximum or exact connectivity on links. However, we ignore this problem here and concentrate on modelling the algorithm.

Figures 10.28 and 10.29 describe steps 2 and 3 in our algorithm. Figure 10.29 is the interesting rule, using absence to encode the inference that if a square containing **7** is the only square in a particular row which can contain **7** then it must contain **7**.

The instantiations associated with these rules are fairly trivial. We never copy parameters and

**Figure 10.29:** *Filling a square through inference*

only one set of rules discards a parameter. There are a large number of reaction rules but after all, we are describing a case-based algorithm.

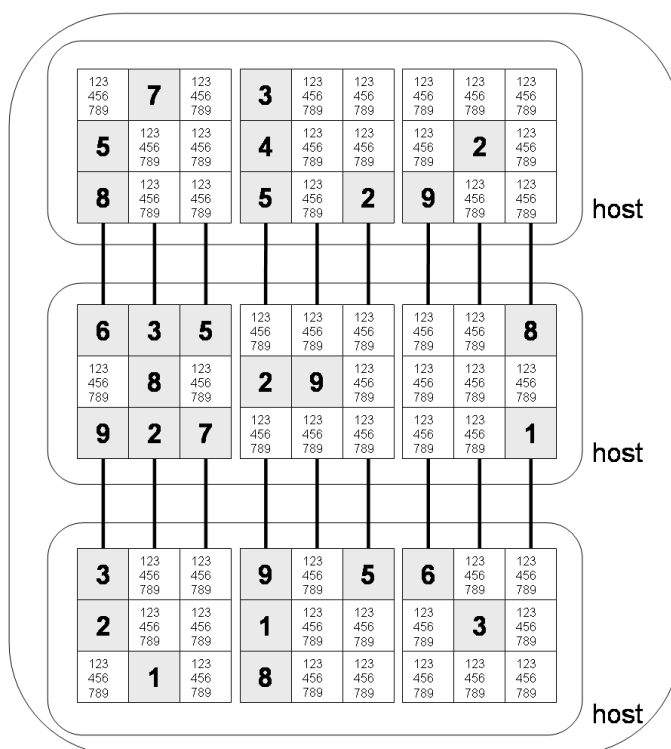
The final ingredient for our encoding is the translation from a Sudoku game to a bigraph. We label, in some order, the rows of a Sudoku puzzle from  $x_1$  to  $x_9$ , the columns from  $y_1$  to  $y_9$ , and the blocks as  $z_1$  to  $z_9$ . The square at row  $x$ , column  $y$  which is in block  $z$  is represented by  $\text{square}_{x,y,z}$ .

$$\begin{aligned} \llbracket \text{Puzzle} \rrbracket &= \llbracket \text{square}_{1,1,1} \rrbracket \parallel \llbracket \text{square}_{1,1,2} \rrbracket \parallel \cdots \parallel \llbracket \text{square}_{9,9,8} \rrbracket \parallel \llbracket \text{square}_{9,9,9} \rrbracket \\ \llbracket \text{square}_{x_i,y_j,z_k} \rrbracket &= \begin{cases} \mathbf{sq}_{x_iy_jz_k} \circ (1|2|\cdots|9) & \text{if square is unfilled} \\ \mathbf{sq}_{x_iy_jz_k} \circ i & \text{if square contains } i, i \in \{1, \dots, 9\} \end{cases} \end{aligned}$$

Note that the translation of a Sudoku puzzle into bigraphs does not need to colocate all the squares. Furthermore, the reaction rules are wide. This allows us to distribute a puzzle across multiple hosts as in Figure 10.30 (we omit most of the links for clarity). The wide reaction rules allow cross-border reactions so that each host ends up with a local piece of the solved puzzle whilst no one host knows the complete solution (or has to).

The reaction rules will then supply a solution to a puzzle if: (i) it has a unique solution; (ii) it does not require reasoning outside of our algorithm; and (iii) we close the names in the encoding but do not merge them.

The system is strongly normalising and seems locally confluent which suggests a simple proof of confluence by Newman's lemma [117].

Figure 10.30: *Distributed Sudoku*

## 10.6 Semi-structured data

Cardelli observed that research in semi-structured data and mobile computation, which respectively describe data and computation in distributed systems, share technical similarities and proposed a spatial logic for describing semi-structured tree-like data [28]. The description of semi-structured data in terms of spatial logic has been studied considerably since then [29, 44, 30]. Of most relevance to us is the spatial logics for bigraphs proposed by Conforti, Macedonio, and Sassone and applied to this problem [39, 38].

Semi-structured data has emerged as an important means to transfer information across a distributed system. In this section, we demonstrate how kind sorting allows the modelling of semi-structured data such as XML data with document order and BibTeX entries. As bigraphs are meant to model distributed systems, this gives a clear application of the sorting and solves the problem mentioned by Conforti, Macedonio, and Sassone that since place graphs do not order their children, pure bigraphs can only model XML data with no document order.

### 10.6.1 XML documents

XML [148] is one of the standard description languages used in distributed systems. Its tree-like structure lends itself well to both spatial logic and a bigraphical encoding where linking is used to name elements (as in an ID attribute) and refer to other data (as in an IDREF). XML data are modelled by ground bigraphs; XML contexts are modelled by non-ground bigraphs.

Conforti, Macedonio, and Sassone give an encoding of XML documents with no document order as ground bigraphs [39, 38]. Figure 10.31 contains our version of the encoding which is identical except that multi-nodes are used to order the elements of a tag. This allows us to correctly model the document order.

We briefly summarise the encoding, referring the reader to the original work for a full explanation. The signature for the encoding is made up from  $K_{val}(v)$  controls which encoding a value  $e.g.$   $v$ , and  $K_{tag}(t, 1 + k + p, n)$  controls representing tags  $t$ . A  $K_{val}(v)_a$  ion connects the single port of the control to an attribute name  $a$ . A  $K_{tag}(t, 1 + k + p, 1 + n)$  control has  $1 + k + p$  ports where the first represents the ID of the tag, the next  $k$  represent links (like IDREFs), and the final  $p$  ports represent value attributes. The  $k$  link-ports connect to names in the model which represent names of other tags; this is reflected in the encoding by the  $\vec{u}$  links of the  $K_{tag}(t, 1 + k + p, 1 + n)_{u, \vec{u}, \vec{b}}$  ion. The value attributes are represented by controls  $\llbracket \vec{v} \rrbracket_{\vec{b}}$  and stored inside the tag node. The  $1 + n$  is our addition; we model the tag node as a multi-node with  $1 + n$  compartments. Therefore, we also omit the merge bigraph in the encoding of an XML tree. The first compartment stores all the value attributes with no placing order (the order is given by the arity of the tag control). The final  $n$  compartments store the encodings of the children tags.

The remainder of the encoding involves linking constructs – renamings and closures – to ensure the translation is valid with respect to the tensoring conditions (disjointness of names) and to close the links between value attributes and tags.

We have omitted describing the kind relation of the signature; this falls out of the XML schema as does the choice of which of the kind signatures of Sections 3.4–3.6 best suits the model.

### 10.6.2 BibTeX files

Similarly to XML, the BibTeX file format [87, 127] is essentially an unordered forest of unordered trees with linkage in the form of citation keys and cross-references. BibTeX files can therefore be naturally expressed as bigraphs. A simple kind sorting over an appropriate signature allows us to define s-categories whose bigraphs represent the prime and parallel (*i.e.* unordered) composition of BibTeX entries.



|                                           |                                                                                                                                                                                                                                                                                 |                                              |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| $\llbracket v \rrbracket$                 | $\stackrel{\text{def}}{=} K_{val}(v)$                                                                                                                                                                                                                                           | value                                        |
| $\llbracket v \rrbracket_a$               | $\stackrel{\text{def}}{=} K_{val}(v)_a$                                                                                                                                                                                                                                         | value linked to an attribute name $a$        |
| $\llbracket \vec{v} \rrbracket_{\vec{b}}$ | $\stackrel{\text{def}}{=} \llbracket v_1 \rrbracket_{b_1} \otimes \cdots \otimes \llbracket v_n \rrbracket_{b_n}$                                                                                                                                                               | with $ \vec{v}  =  \vec{b}  = n$             |
| $\llbracket 0 \rrbracket$                 | $\stackrel{\text{def}}{=} 1$                                                                                                                                                                                                                                                    | empty tree                                   |
| $\llbracket T \rrbracket$                 | $\stackrel{\text{def}}{=} / \vec{a} \circ \sigma \circ K_{tag}(t, 1 + k + p, 1 + n)_{u, \vec{a}, \vec{b}} \circ (\llbracket \vec{v} \rrbracket_{\vec{b}} \otimes (\alpha_1 \circ \llbracket T_1 \rrbracket) \otimes \cdots \otimes (\alpha_n \circ \llbracket T_n \rrbracket))$ |                                              |
| where                                     | $T = \langle t, \text{ID} = u, \vec{a} = \vec{u}, \vec{b} = \vec{v} \rangle T_1, \dots, T_n \langle /t \rangle$                                                                                                                                                                 | XML tree                                     |
|                                           | $\vec{a} = a_1 \dots a_k$                                                                                                                                                                                                                                                       | link attributes                              |
|                                           | $\vec{u} = u_1 \dots u_k$                                                                                                                                                                                                                                                       | names                                        |
|                                           | $\vec{b} = b_1 \dots b_p$                                                                                                                                                                                                                                                       | value attributes                             |
|                                           | $\vec{v} = v_1 \dots v_p$                                                                                                                                                                                                                                                       | values                                       |
|                                           | $\alpha_i$                                                                                                                                                                                                                                                                      | renaming the names of $T_i$ into fresh names |
|                                           | $\sigma = \alpha_1^{-1} \cup \cdots \cup \alpha_n^{-1}$                                                                                                                                                                                                                         | the union of the inverse renamings           |
|                                           | $/ \vec{a} \stackrel{\text{def}}{=} / a_1 \otimes \cdots \otimes a_p$                                                                                                                                                                                                           | closure of the names in $\vec{a}$            |

**Figure 10.31:** XML documents as ground kind sorted bigraphs

Figure 10.32 depicts an appropriate kind relation with capacities for modelling BibTeX files. We represent string values with a single atomic control **val** for ease of presentation. As BibTeX entries have optional components, we use a kind signature with rigid min-max capacities. The **phdthesis** entry type has four required fields and four optional fields, represented by the minimum and maximum capacities and the kind function. The **authors** field can contain multiple authors, each of which contains one multi-node which contains an ordered list of  $n$  names in order. We assign an arity of one to each entry type (*e.g.* article, book, *etc.*), used to link to a citation key represented as an outer name. The **crossref** control, not shown in the diagram, also has arity one, allowing **crossref** nodes to link to citation keys.

In order to reflect the BibTeX file format, the model requires a link sorting so that BibTeX entries have unique citation keys. Leifer and Milner's safe many-one sorting [92] seems appropriate, where entry type ports are assigned sort **s** and **crossref** ports are assigned sort **t**. This system could then be refined by considering the subcategory where all names have sort **s** and where no **s**-port has a closed link.

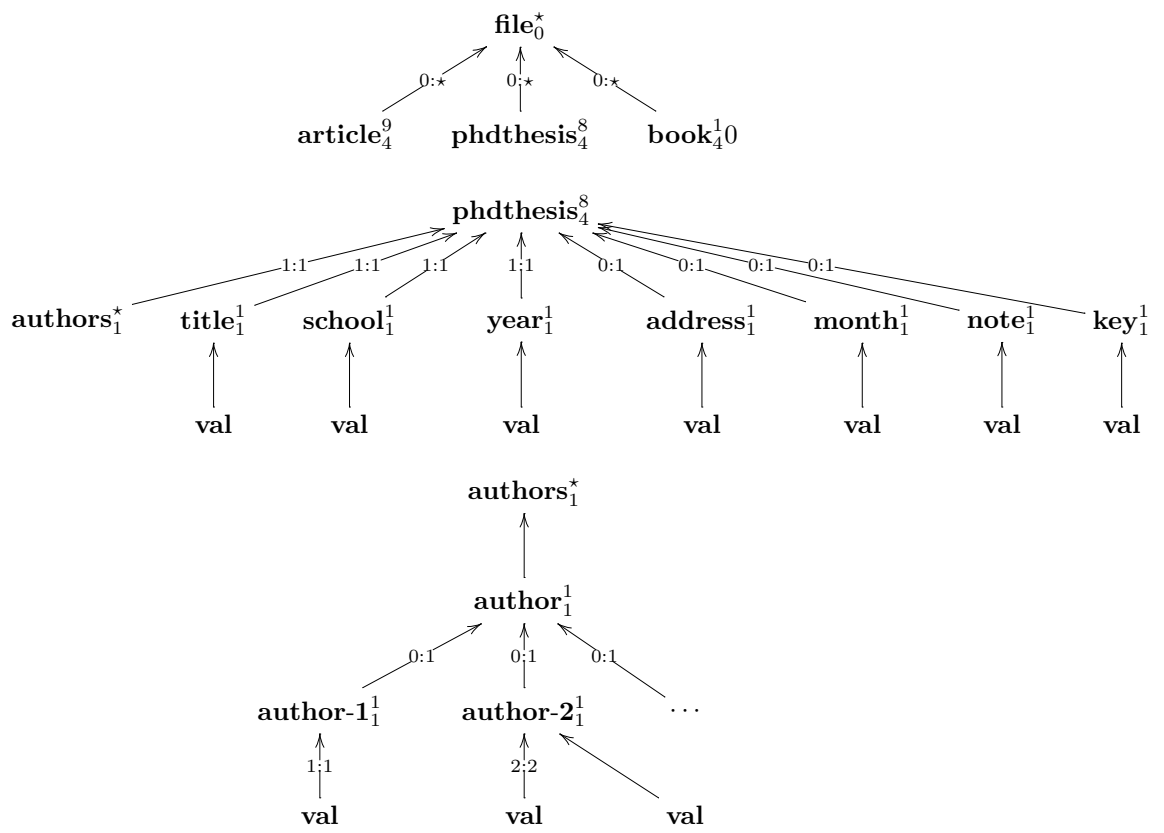


Figure 10.32: An example of a kind relation for BibTeX files

## 10.7 Conclusions

This chapter presented applications for kind sorting including: i) modelling parametric reaction rules with simple preconditions; ii) refining models of context-aware systems; iii) modelling basic flow control allowing some simple algorithms; and iv) adding more order to encodings of semi-structured data commonly used in distributed systems. In the next chapter, we will demonstrate how kind sorting allows us to model some basic typed  $\lambda$ -calculi.

### 10.7.1 Related and further work

We have already discussed some connections between bigraphs and spatial logics in Chapter 3. In particular, we believe it is worthwhile to consider generalising the bigraphical logic BiLog [39] to kind sorted bigraphs – the kind sorting of sites immediately suggests logical statement such as “no  $K$ -node can be placed here” as well as describing preconditions for reaction rules. This expressiveness is of course not limited to kind sortings; different place sortings and link sortings could describe other logical properties and reactive preconditions. This study requires an axiomatisation of kind sortings which we believe is straightforward.

An early motivation behind kind sortings was the author’s attempt to model The Confederation of the Isles formal methods scenario defined by my supervisor [96]. Attempting to model this example in bigraphs required us to add hierarchy and capacity to the place graph structure. Such models also require that pre- and post-conditions can be described for reconfigurations of the system. We have demonstrated that kind rules allows particular pre-conditions to be expressed; it does not seem unreasonable to conjecture that more complicated sorting could be used to allow more complex conditions to be specified.

Most Brss in the current literature model various calculi. This is important but it is also important to gain intuitions as to how bigraphs can be used to model different sorts of systems. Plato-graphical sorting allows the modelling of context-aware systems where subsystems have different views of the world [15]. Debois and Damgaard have presented examples of Brss which model finite automata, Conway’s Game of Life, and combinatory logic [47].

# Chapter 11

## Models of $\lambda$ -calculi

*This machine will will not communicate*

*These thoughts and the strain I am under*

*Street Spirit (fade out) – Radiohead*

In this chapter, we consider typed versions of  $\lambda$ BIG, Milner’s bigraphical model of the  $\lambda$ -calculus with explicit substitutions. The typing is modelled using safe subcategories of kind sortings. While this may not be the ideal way to model typed  $\lambda$ -calculi, it does demonstrate the expressiveness of our sortings.

We first recall the bigraphical reactive system  $\lambda$ BIG. The Brs is based on  $\Lambda_{\text{sub}}$  and reaction in the former corresponds to reduction in the latter. This allows us to reason about the reaction relation using results from Part II of the dissertation.

We use the same sorting scheme, depicted in Figure 11.1, to model untyped, simply typed, and intersection typed  $\lambda$ BIG. More precisely, our sorted bigraphs model *type derivations* in the respective type systems for  $\Lambda_{\text{sub}}$ . The steps in the scheme are:

1. extend the Brs  $\lambda$ BIG with ‘typed’ versions of the controls and reaction rules to define an s-category  $\lambda$ BIG<sup>+</sup>;
2. compose a safe subcategory (*i.e.* the functor is safe) of a kind sorting of  $\lambda$ BIG<sup>+</sup> with the  $\sigma\lambda\nu\delta$ -sorting (Section 11.2) to remove bigraphs whose structure do not reflect the  $\Lambda_{\text{sub}}$  grammar;

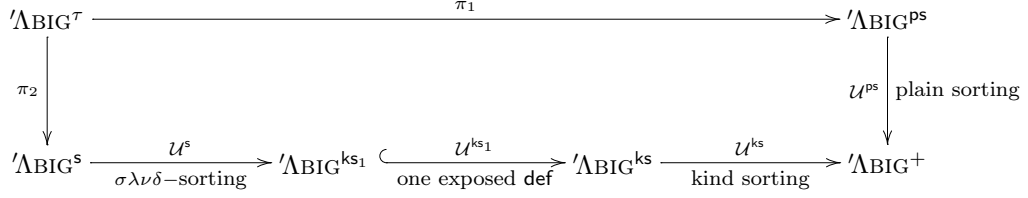


**Figure 11.1:** *Sorting scheme for bigraphical models of  $\lambda$ -calculus*

3. apply a plain sorting to  $\Lambda\text{BIG}^{+1}$ ;
4. take the pullback of these two sortings.

This defines an s-category such that: i) a static correspondence exists between tuples of type derivations in the typed  $\Lambda_{\text{sub}}$  and a large subsystem of the Brs; and ii) all bigraphs model parallel compositions of  $\Lambda_{\text{sub}}$ -terms and bodies of substitution.

Before introducing our models, we define a sorting for Milner's  $\Lambda\text{BIG}$  model which, combined with kind sorting, removes bigraphs which are not encodings of  $\Lambda_{\text{sub}}$  terms. This will allow us to better model the  $\lambda$ -calculus up to  $\alpha$ -equivalence.

Our first model is  $\Lambda\text{BIG}^{\text{ut}}$ , or untyped  $\Lambda\text{BIG}$ , which removes *most* of those bigraphs which are not parallel compositions of encodings of  $\Lambda_{\text{sub}}$  terms<sup>2</sup>. This may be seen as modelling type derivations where the single typing rule allows any term as an axiom.

Next, we use a fully partitioned kind sorting to model  $\Lambda\text{BIG}^{\rightarrow}$ , simply typed  $\Lambda\text{BIG}$ . The partitioning reflects the intuition that types in the type system are unrelated to each other. We introduce a functor from  $\Lambda\text{BIG}^{\rightarrow}$  (and later  $\Lambda\text{BIG}^{\cap}$ ) to  $\Lambda\text{BIG}^{\text{ut}}$  which recovers the bigraphical encoding of untyped  $\Lambda_{\text{sub}}$  terms. This allows us to reason about normalisation and confluence properties for models of typed  $\Lambda_{\text{sub}}$  based on the untyped calculus.

Finally, we present a model of  $\Lambda\text{BIG}^{\cap}$ ,  $\Lambda\text{BIG}$  with intersection types. We use a bounded complete and controlled meet s-category with the intuition that a meet s-category allows a semantic interpretation (we model types as sets of controls) of the subtyping of intersection types via set intersection and that a bounded complete s-category corresponds semantically to the bounded complete property of the type preorder<sup>3</sup>.

<sup>1</sup>In the untyped case, the plain sorted s-category has a single name type. Therefore, the sorting functor is an isomorphism of s-categories and the composition on the bottom line suffices to sort  $\Lambda\text{BIG}^{+}$ .

<sup>2</sup>We allow encodings of bodies of substitution for the  $\rightarrow_C$ -rule. These still present problems which we discuss.

<sup>3</sup>This property is proven in Section 11.5.1.

This last model of a typed λ-calculus is particularly interesting for one reason. The intersection type system we use characterises the set of strongly normalising terms of  $\Lambda_{\text{sub}}$ , which is a conservative extension of the λ-calculus. Hence,  $'\Lambda\text{BIG}^\cap$  models all (and besides the extra bodies of substitution, only) finite parallel compositions of strongly normalising  $\Lambda_{\text{sub}}$  terms; it is a model of all parallel computations, possibly mid-computation, which terminate.

## 11.1 'ΛBIG

'ΛBIG, Milner's encoding of the λ-calculus as a local bigraphical reactive system [107, 111], is both a further exploration of the notion of name-scoping and a starting point for studying confluence in the theory of bigraphs. With 'ΛBIG, Milner also introduced the notion of non-local substitution in bigraphs which we indirectly investigated in Part II with encouraging results.

### 11.1.1 The encoding of $\Lambda_{\text{sub}}$ into 'ΛBIG

**Definition 11.1** ('ΛBIG signature). *The set of controls of 'ΛBIG is:*

$$\{\text{lam} : 1, \text{app} : 0, \text{var} : 1, \text{sub} : 1, \text{def} : 1\}.$$

*The controls lam and sub are binding, the other controls are nonbinding. var is the only atomic control and all other controls are active.*

'ΛBIG does not model the λ-calculus itself but rather encodes the explicit substitution calculus  $\Lambda_{\text{sub}}$ , which is a conservative extension of the λ-calculus (see Section 8.1).

**Definition 11.2** ( $\Lambda_{\text{X}}$ -terms into bigraphs [110]).

$$\begin{aligned} \llbracket x \rrbracket_{X \uplus x} &\stackrel{\text{def}}{=} \text{var}_x \oplus X \\ \llbracket \lambda x. t \rrbracket_X &\stackrel{\text{def}}{=} (\text{lam}_{(x)} \oplus \text{id}_X) (\llbracket t \rrbracket_{X \uplus x}) \\ \llbracket t \ u \rrbracket_X &\stackrel{\text{def}}{=} (\text{app} \oplus (\text{id}_X \mid \text{id}_X)) (\llbracket t \rrbracket_X \parallel \llbracket u \rrbracket_X) \\ \llbracket t[x/u] \rrbracket_X &\stackrel{\text{def}}{=} (\text{sub}_{(x)} \oplus \text{id}_X) (\llbracket t \rrbracket_{X \uplus x} \mid (\text{def}_x \oplus \text{id}_X) \llbracket u \rrbracket_X). \end{aligned}$$

The encoding models  $\Lambda_{\text{sub}}$  terms up to  $\alpha$ -equivalence. Closures  $t[x/u]$  are modelled in 'ΛBIG using two controls, **def** and **sub**. The **def** node contains the body of substitution of the closure and the **sub** node models the binding. This separation of body from closure allows non-local substitution to be modelled.

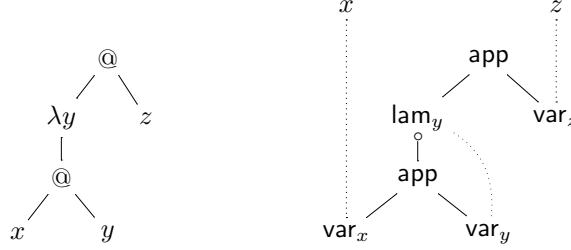


**Figure 11.2:** Abstract syntax tree for  $(\lambda y.xy)z$  and a representation of the corresponding prime bigraph

The structure of an encoded term could be considered as the usual abstract syntax tree representation augmented with an explicit linking structure for binding names. See Figure 11.2 for example.

The index  $X$  of the encoding  $\llbracket t \rrbracket_X$  is a set which must at least contain the free variables of  $t$  in  $\Lambda_{\text{sub}}$ . This is necessary for technical reasons; reaction in Brss preserve interfaces *i.e.* the reactum of a rule has the same set of outer names as the redex. For example, the garbage collection reduction  $\llbracket x[y/z] \rrbracket_{\{x,z\}} \rightarrow_D \llbracket x \rrbracket_{\{x,z\}}$  preserves the free name  $z$ .

The reaction rules of 'ABIG are those in Figure 11.6 if we forget the sorts. The  $\rightarrow_C$  reaction demonstrates how non-local substitution uses the link graph to replace variables with bodies of substitution, implicitly allowing full composition of substitutions.

In Appendix C.3 we consider similarities between 'ABIG and both  $\lambda_{\text{lr}}$  and the  $\pi$ -calculus encoding of the lazy  $\lambda$ -calculus. We also compare the reduction strategy of 'ABIG with previous encodings of the  $\lambda$ -calculi in process calculi.

### 11.1.2 Dynamic properties

The reaction rules match the reduction relation of  $\Lambda_{\text{sub}}$ . Milner stated the following dynamic correspondence between  $\Lambda_{\text{sub}}$  terms and the image of the encoding [107].

**Proposition 11.3** (reaction matches reduction).  $\llbracket t \rrbracket_X \rightarrow g$  if and only if  $t \rightarrow_{\text{bcgc}} t'$  for some  $t'$  such that  $\llbracket t' \rrbracket_X \simeq g$ .

As a corollary, we can transfer our proofs of confluence and normalisation from  $\Lambda_{\text{sub}}$  to 'ABIG.

**Corollary 11.4.** 'ABIG can simulate  $\beta$ -reduction step-by-step, is confluent on encodings of  $\Lambda_{\text{sub}}$  terms, and is strongly normalising on encodings of  $\Lambda_{\text{sub}}$  terms typable by in the type system of Section 9.4.

**Corollary 11.5** (PSN for the image of the encoding). *Encodings of pure terms which are strongly normalising for  $\beta$ -reduction are strongly normalising.*

## 11.2 Sorting out $\Lambda$ BIG

$\Lambda$ BIG encodes  $\Lambda_{\text{sub}}$  terms up to  $\alpha$ -equivalence. However, not all bigraphs in  $\Lambda$ BIG are encodings of  $\Lambda_{\text{sub}}$  terms. In the following sections, we present kind and plain sortings to remove most of these bigraphs. In this section we present a different sorting which removes most remaining reprobates that those sortings cannot. First, we identify the bigraphs we wish to prohibit. Next, we define this further sorting and prove it safe. Finally, we combine the sortings using the methods discussed in the Section 6.3 to define a safe sorting which removes junk bigraphs from  $\Lambda$ BIG.

### 11.2.1 The good, the bad, and the ugly

We know which bigraphs we do not want to remove; we want to keep all encodings of  $\Lambda_{\text{sub}}$  terms. It is therefore important that the sorting we use is satisfied by the encoding.

The grammar of  $\Lambda_{\text{sub}}$  has implicit capacities; variables contain no subterms, abstractions contain exactly one, and applications and substitutions contain exactly two (which are ordered). We can remove bigraphs in  $\Lambda$ BIG which do not obey this grammar by applying a kind sorting with semi-rigid maximum capacities. Note that it does not make sense for two exposed **def**-nodes in a bigraph  $F$  to be siblings as no kind sorted context  $G$  exists such that  $G \circ F$  is sorted. Therefore, we consider a subcategory of the kind sorted s-category where only one **def**-node can be exposed per root. Our treatment of (basic) type systems also requires a simple linking condition where all points in a link have equal sort. A plain sorting can guarantee this condition.

The kind sorting ensures that all  $\Lambda$ BIG bigraphs have the correct place graph structure but it cannot remove the remaining junk bigraphs; these involve both the nesting and the linking structure. The grammar of  $\Lambda_{\text{sub}}$  states that: variables  $x$  are ‘linked’ to a free name; the abstraction  $\lambda x.t$  binds free occurrences of variables  $x$  in  $t$ ; and the closure  $t[x/u]$  binds free occurrences of variables  $x$  only in  $t$ . These linking constraints can be broken in  $\Lambda$ BIG when:

- a **sub**-node binds a **def** which is not its body of substitution;
- a **sub**-node does not bind its body of substitution;
- a **sub**-node binds a **var** which is contained in its body of substitution;
- a **lam**-node binds a **def** port;



- two **def** ports are linked together;
- a **def** port is linked to a **var** port in the same region;

where we call the **def**-node directly under the right hand side of a **sub**-node the *body of substitution of the sub-node*. Some of these cases arise because a closure  $t[x/u]$  is modelled with a **sub**-node and a **def**-node. On the one hand, this seems a good solution as **def**-nodes enable the modelling of non-local substitution. On the other hand, pure and local bigraphs do not force controls to be related to each other so the **def**-nodes are not ‘glued’ to their **sub**-nodes. The kind sorting ensures that a **def**-nodes is localised to a **sub**-node, but it cannot ensure that both nodes are linked properly. The remaining cases cannot be solved using a kind sorting either. We therefore define a sorting particular to  $\Lambda$ BIG. This will then be composed with the kind sorting.

### 11.2.2 $\Lambda$ BIG $\sigma\lambda\nu\delta$ -sorting

The following sorting removes those bigraphs from  $\Lambda\text{BIG}^{\text{ks}_1}$  where the linking does not reflect the  $\Lambda_{\text{sub}}$  structure.

**Definition 11.6** ( $\Lambda$ BIG  $\sigma\lambda\nu\delta$ -signature). *The  $\Lambda$ BIG  $\sigma\lambda\nu\delta$ -signature assigns sorts to the ports of  $\Lambda$ BIG controls as follows:*

$$\text{lam} \mapsto \lambda, \text{ var} \mapsto \nu, \text{ sub} \mapsto \sigma, \text{ def} \mapsto \delta.$$

We call the port of a **lam**-node a  $\lambda$ -port and use similar terminology for the other controls.

This is a sorting *scheme*. We will apply it to different signatures with ‘typed’ versions of the  $\Lambda$ BIG controls in the obvious manner. The sorting only allows  $\nu$  and  $\delta$  as interface sorts. We call a name with sort  $\nu$  (resp.  $\delta$ ) a  $\nu$ -name (resp.  $\delta$ -name). If a port  $p$  or name  $x$  has sort  $\theta$  then we write  $p : \theta$  or  $x : \theta$  respectively.

**Definition 11.7** ( $\sigma\lambda\nu\delta$ -sorting). *The  $\sigma\lambda\nu\delta$ -sorting  $\Sigma = (\mathcal{K}, \{\nu, \delta\}, \Phi)$  over the  $\Lambda$ BIG  $\sigma\lambda\nu\delta$ -signature  $\mathcal{K}$  assigns sorts  $\{\nu, \delta\}$  to names of interfaces and has the condition  $\Phi$  which requires for all bigraphs  $G$  of  $\Lambda\text{BIG}^{\text{ks}_1}$  that:*

- L1** each outer  $\nu$ -name links arbitrarily many inner  $\nu$ -names and  $\nu$ -ports;
- L2** each outer  $\delta$ -name links arbitrarily many inner  $\nu$ -names and  $\nu$ -ports and exactly one  $\delta$ -point;
- L3** each  $\sigma$ -port binds arbitrarily many inner  $\nu$ -names and  $\nu$ -ports and exactly one  $\delta$ -point;

**L4** each  $\lambda$ -port binds arbitrarily many inner  $\nu$ -names and  $\nu$ -ports;

**P1** if  $v_1 <^2 v_2$  with  $\text{ctrl}(v_1) = \text{def}$  and  $\text{ctrl}(v_2) = \text{sub}$  then the port  $p_{v_2}$  of  $v_2$  binds the port  $p_{v_1}$  of  $v_1$  i.e.  $(p_{v_1}, p_{v_2}) \in \text{loc}_G$ ;

**P2** if  $v_1 < v_2 <^1 v_3$  with  $\text{ctrl}(v_1) = \text{var}$ ,  $\text{ctrl}(v_2) = \text{D}$ , and  $\text{ctrl}(v_3) = \text{sub}$  then the port  $p_{v_3}$  of  $v_3$  does not bind the port  $p_{v_1}$  of  $v_1$  i.e.  $(p_{v_1}, p_{v_3}) \notin \text{loc}_G$ ;

**P3** if  $s <^1 v_1 <^1 v_2$  with  $\text{ctrl}(v_1) = \text{D}$  and  $\text{ctrl}(v_2) = \text{sub}$  then there exists an inner name  $x : \delta$  located at  $s$  which is the only name located at  $s$  to be linked to the port  $p_{v_2}$  of  $v_2$  i.e.  $(s, x : \sigma) \in \text{loc}_G$ ,  $\text{link}(x) = p_{v_2}$ ,  $\text{link}(y) \neq p_{v_2}$  when  $y \in \text{loc}_G(s)$  with  $y \neq x$ ;

**P4** no inner  $\delta$ -names located under a  $\text{def}$ -node can link to outer  $\delta$ -names;

**P5** an outer  $\delta$ -name cannot link an inner  $\delta$ -name or  $\delta$ -port to any other points located in any of the regions where the inner name or port is located<sup>4</sup>;

**P6** if  $s < v_1 <^1 v_2$  with  $\text{ctrl}(v_1) = \text{D}$  and  $\text{ctrl}(v_2) = \text{sub}$  then no inner  $\nu$ -names located at  $s$  are bound by the port  $p$  of  $v_2$  i.e. if  $\text{link}(x) = p$  and  $(s, x) \in \text{loc}_G$  then  $x : \delta$ ;

**P7** no inner  $\delta$ -names are located under a  $\text{U}$ -node unless they are linked to a  $\sigma$ -port;

where  $s$  is a site.

The linking rules L1-L4 specify the correct cardinalities for the linking structure. The placing rules P1-P7 help ensure that: i) if a  $\text{def}$ -node lies just beneath a  $\text{sub}$ -node then their ports are linked, and ii) no other node under the right hand side of the  $\text{sub}$ -node is bound by the  $\text{sub}$ -node.

Composition is defined in the sorted s-category when the underlying composition in  $\lambda\text{BIG}^{\text{ks}_1}$  is defined (and hence when the underlying composition of local bigraphs in  $\lambda\text{BIG}^+$  is defined).

**Proposition 11.8** (composition respects  $\sigma\lambda\nu\delta$ -sorting). *If  $A : H \rightarrow I$  and  $B : I \rightarrow J$  are sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.*

*Proof.* See Appendix A.6, Proposition C.2. □

The identities and tensor product respect the sorting. Therefore, the sorted bigraphs form an s-category  $\lambda\text{BIG}^s$  with interfaces  $(m, \theta_1, X, \theta_2)$ , where  $\theta_1$  is a kind sort and  $\theta_2 \in \{\lambda, \sigma\}$ , and  $\sigma\lambda\nu\delta$ -sorted bigraphs as arrows. The sorting functor  $\mathcal{U}^s : \lambda\text{BIG}^s \rightarrow \lambda\text{BIG}^{\text{ks}_1}$  forgets the name sorts of interfaces and is faithful. The parametric reaction rules of  $\lambda\text{BIG}^+$  can be sorted by assigning sort

<sup>4</sup>In the first case, this includes other locations (sites) of the inner  $\delta$ -name.

$\nu$  to the inner name of the substitution generation rule and  $\delta$  to the outer name of the substitution rule. The encodings of (typed)  $\lambda$ -terms in Chapter 11 also respect the sorting.

Therefore, the sorting yields a Brs which removes most of and only the junk bigraphs. Furthermore,  $\mathcal{U}^s$  creates RPOs and reflects pushouts (see Appendix A.6, Propositions C.7 and C.4 respectively).

### 11.3 Modelling the untyped $\lambda$ -calculus

Figure 11.1 depicts our sorting scheme for models of  $\lambda$ -calculi with a basic type system where we encode *type derivations* rather than terms<sup>5</sup>. In order to apply this scheme generically to untyped  $\Lambda_{\text{sub}}$ , we consider it as a calculus with a single type  $*$  and the trivial axiom in Figure 11.3 which is true for all terms. However, we will not explicitly use this type or the axiom.

#### 11.3.1 The sorting for $\Lambda\text{BIG}^{\text{ut}}$

We begin by adding kinds with maximum capacities to the signature to match the hierarchical structure of the grammar of  $\Lambda_{\text{sub}}$ . We model applications and closures with multi-nodes to allow their subterms to be ordered.

**Definition 11.9** ( $\mathcal{K}^*$ ). *The set  $\mathcal{K}^*$  is defined as  $\{\text{var}, \text{app}, \text{sub}, \text{lam}\}$ .*

**Definition 11.10** ( $\Lambda\text{BIG}^{\text{ks}}$  (kind) signature). *The kind signature  $\mathcal{K}^{\text{ut}}$  with maximum capacities for  $\Lambda\text{BIG}^{\text{ks}}$  is defined in Figure 11.4 where  $0^{\mathcal{K}^*}$  denotes the constant function  $0^{\mathcal{K}^*} : \mathcal{K} \rightarrow \{0\}$  and  $1^{\mathcal{K}^*}$  denotes the constant function  $1^{\mathcal{K}^*} : \mathcal{K} \rightarrow \{1\}$ . It adds kinds to the pure signature and augments it with four active, invisible controls 1, 2, U, and D of arity zero. The 1 and 2 controls are associated with the **app** control and U and D controls are associated with the **sub** control.*

A graphical representation of the signature (which we prefer) is presented in Figure 11.4. To avoid confusing the diagram, we have omitted the labels on the arrows, all of which are ‘1’ except

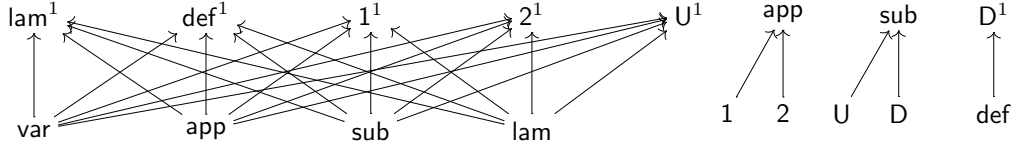
<sup>5</sup>Each control, name, and port has an associated type (see Figure 11.10) so that the link sorting, place sorting, and nesting structure resemble proof trees; moreover, the encodings maps derivations, not terms, to bigraphs.

$$\boxed{\frac{\text{FV}(t) \subseteq \Gamma \quad \text{BV}(t) \cap \Gamma = \emptyset}{\Gamma \vdash t : *} \quad (\text{triv})}$$

**Figure 11.3:** The type system for untyped  $\Lambda\text{BIG}$

| Control | $kind_{inv}$ | $kind$           | $cpc$ |
|---------|--------------|------------------|-------|
| var     | $\emptyset$  | $0\mathcal{K}^*$ | 0     |
| lam     | $\emptyset$  | $1\mathcal{K}^*$ | 1     |
| app     | $\{1, 2\}$   | $0\mathcal{K}^*$ | 0     |
| sub     | $\{U, D\}$   | $0\mathcal{K}^*$ | 0     |
| def     | $\emptyset$  | $1\mathcal{K}^*$ | 1     |

| Control | $kind_{inv}$ | $kind$           | $cpc$ |
|---------|--------------|------------------|-------|
| 1       | $\emptyset$  | $1\mathcal{K}^*$ | 1     |
| 2       | $\emptyset$  | $1\mathcal{K}^*$ | 1     |
| U       | $\emptyset$  | $1\mathcal{K}^*$ | 1     |
| D       | $\emptyset$  | $1 \text{ def}$  | 1     |

(a) Kind sorting for  $\lambda\text{BIG}$  with semi-rigid maximum capacities(b) Graphical representation of the  $\lambda\text{BIG}$  kind sorting**Figure 11.4:** Two presentations of the kind sorting for  $\lambda\text{BIG}$ 

the arrows originating from invisible controls (which have no label). The ions corresponding to the sorting are depicted in Figure 11.5. The interface sorts for the ions (and, later, the reaction rules) are the only sorts possible from the fully partitioned subcategory we take over the sorting.

**Definition 11.11** ( $\lambda\text{BIG}^+$ ,  $\lambda\text{BIG}^{ks}$ ). *The pure s-category  $\lambda\text{BIG}^+$  consists of controls  $\mathcal{K}^{ut}$  and the reaction rules of Figure 11.6, forgetting the sorts. The kind sorted Brs with semi-rigid maximum capacities  $\lambda\text{BIG}^{ks}$  is the reactive system over the fully partitioned s-category of  $\mathcal{B}\text{IG}_h(\Sigma_K)$  with the signature  $\mathcal{K}^{ut}$ , the set of interface kinds  $\{\mathcal{K}^*, \{\text{def}\}\}$ , and where the reaction rules are as depicted in Figure 11.6.*

We will require that roots of bigraphs have at most one exposed **def**-node. From the point of view of the model, this is fine; **def**-nodes and their contents do not represent  $\Lambda_{\text{sub}}$  terms. They only need to be exposed for the  $\rightarrow_C$  reaction rule.

**Definition 11.12** ( $\lambda\text{BIG}^{ks_1}$ ). *The Brs  $\lambda\text{BIG}^{ks_1}$  is defined by taking the largest full subcategory of  $\lambda\text{BIG}^{ks}$  where if  $kind(r)(\text{def}) > 0$  then  $kind(r)(\text{def}) = 1$  and where capacities of places are not zero or  $\odot$ . The reaction rules of  $\lambda\text{BIG}^{ks}$  all satisfy these conditions.*

In other words, a bigraph has at most one exposed **def** node per root and we disallow barren roots and roots with arbitrary capacities (which could be barren also). Note that this forces hardness. The conditions are satisfied by the identities, composition, and tensor product. This subcategory is then composed with the  $\sigma\lambda\nu\delta$ -sorting to complete the bottom line of the sorting scheme.

**Definition 11.13** ( $'\Lambda\text{BIG}^s$ ). *The Brs  $'\Lambda\text{BIG}^s$  is defined by applying the  $\sigma\lambda\nu\delta$ -sorting to  $'\Lambda\text{BIG}^{ks_1}$ .*

**Definition 11.14** ( $'\Lambda\text{BIG}^{ps}$ ). *The Brs  $'\Lambda\text{BIG}^{ps}$  is defined by assigning the sort  $*$  to the ports of controls in  $'\Lambda\text{BIG}^+$  and by enriching the interfaces with the constant sorting with codomain  $*$ .*

In this untyped case, the plain sorting uses a single link type  $*$  and the sorting condition is always satisfied. As the sorting condition is always satisfied, the sorting functor is full. Since the objects of the plain sorted s-category are in bijection with those of  $'\Lambda\text{BIG}^+$  and the sorting functor is also faithful, the plain sorted s-category is isomorphic to  $'\Lambda\text{BIG}^+$  in this case.

Finally, we take the pullback/pairing of the sortings and prove that the dynamic theory is transferred along the sorting.

**Definition 11.15** ( $'\Lambda\text{BIG}^{ut}$ ). *The Brs  $'\Lambda\text{BIG}^{ut} = '\Lambda\text{BIG}^s \times_{'\Lambda\text{BIG}^+} '\Lambda\text{BIG}^{ps}$  is defined as the paired sorting of  $'\Lambda\text{BIG}^s$  and  $'\Lambda\text{BIG}^{ps}$ . The ions and reaction rules of  $'\Lambda\text{BIG}^{ut}$  are sorted as depicted in Figures 11.5 and 11.6 respectively.*

**Proposition 11.16.**  $\mathcal{U}^{ks_1}$  creates RPOs and reflects pushouts.

*Proof.*  $\mathcal{U}^{ks_1}$  is a full subcategory. By Lemmas 5.35 and 5.40, it suffices to show that no place of an RPO interface has a sort  $\theta$  such that  $cpc(\theta) = 0$ ,  $cpc(\theta) = \odot$ , or  $kind(r)(\text{def}) > 1$ .

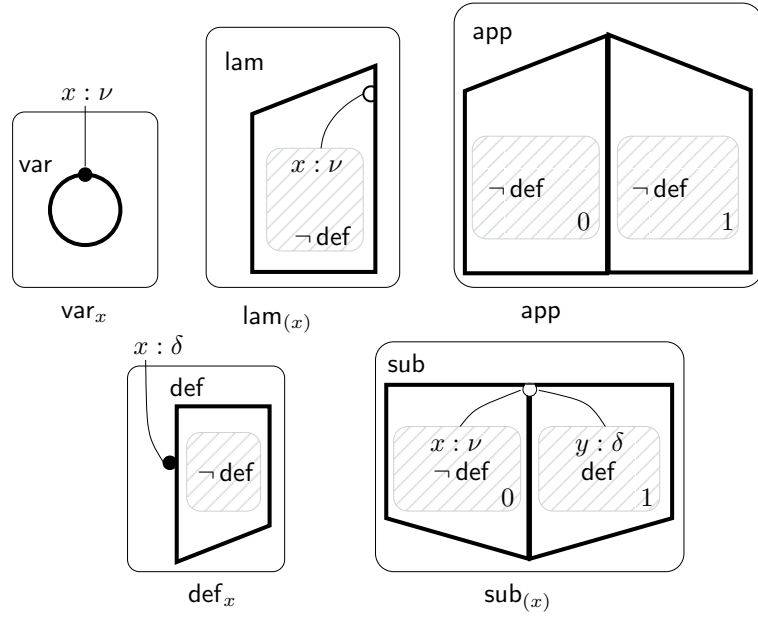
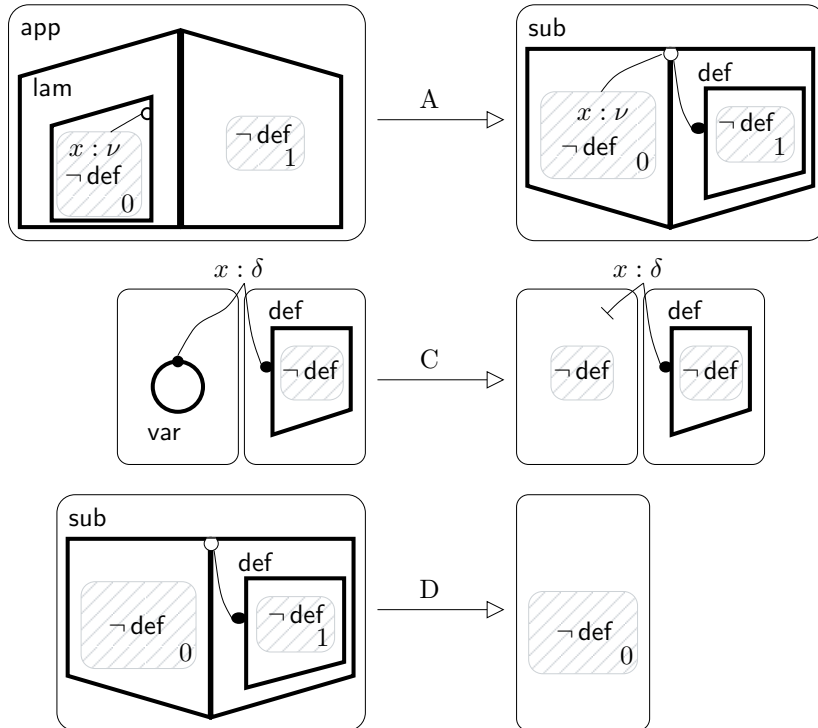
By the RPO construction for kind sorting with semi-rigid capacities, a place will only have capacity 0 or  $\odot$  if a child site in one of the legs of the RPO has the same capacity. These place sorts are not allowed in the subcategory we work in.

Assume a place  $r$  of the RPO interface has  $kind(r)(\text{def}) = n, n > 1$ . This place is a site of the body  $B$  of the RPO. By the kind sorting, we must have  $r <_B t$ , where  $t$  is a root. Therefore,  $kind(t)(\text{def}) \geq n$ . However, this place sort is not allowed in the subcategory we work in.

Therefore, the interface of the RPO is an object of  $\mathcal{U}^{ks_1}$  and the proof follows.  $\square$

**Proposition 11.17.** *The functor  $\mathcal{U}^{ks} \circ \mathcal{U}^{ks_1} \circ \mathcal{U}^s \circ \pi_2$  creates RPOs and reflects pushouts.*

*Proof.* By Propositions 5.36, 5.42, 11.16, C.7, and Corollary C.4, the functors  $\mathcal{U}^{ks}$ ,  $\mathcal{U}^{ks_1}$ , and  $\mathcal{U}^s$  create RPOs and reflect pushouts. Therefore the composition  $\mathcal{U}^{ks} \circ \mathcal{U}^{ks_1} \circ \mathcal{U}^s$  also has these

Figure 11.5: Ion schema for  $\Lambda\text{BIG}^{\text{ut}}$ Figure 11.6: Parametric reaction rule schema for  $\Lambda\text{BIG}^{\text{ut}}$

| Derivation             | Encoding with set $X$ (all variables in $X$ have type $\nu$ )                                                                                                                                                                                              |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $X \vdash x$           | $\text{var}_{x:\nu} \oplus X/\{x:\nu\}$                                                                                                                                                                                                                    |
| $X \vdash \lambda x.t$ | $(\text{lam}_{(x:\lambda)} \oplus \text{id}_X) \llbracket X \uplus x : \nu \vdash t \rrbracket$                                                                                                                                                            |
| $X \vdash t \ u$       | $(\text{app} \oplus (\text{id}_X \mid \text{id}_X))(\llbracket X \vdash t \rrbracket \parallel \llbracket X \vdash u \rrbracket)$                                                                                                                          |
| $X \vdash t[x/u]$      | $(\text{sub}_{(x:\delta)} \oplus (\text{id}_X \mid \text{id}_X))(\{x:\delta/x:\nu \ \bar{y}:\delta\} \circ (\llbracket X \uplus x : \nu \vdash t \rrbracket \parallel (\text{def}_{\bar{y}:\delta} \oplus \text{id}_X) \llbracket X \vdash u \rrbracket))$ |

**Figure 11.7:** Encoding of untyped  $\Lambda_{\text{sub}}$  terms into  $'\Lambda\text{BIG}^{\text{ut}}$ 

properties. By Propositions 6.22 and 6.23, the functor  $\mathcal{U}^{\text{ps}}$  creates RPOs and reflects pushouts. We conclude by Propositions 6.29 and 6.30.  $\square$

**Corollary 11.18.** *Wide bisimilarity over the standard transition system  $\text{ST}$  for  $'\Lambda\text{BIG}^{\text{ut}}$  is a congruence and mono bisimilarity is a congruence for mono contexts.*

*Proof.* By Theorem 5.10.  $\square$

### 11.3.2 Static correspondence

The encoding of  $\Lambda_{\text{sub}}$  into  $'\Lambda\text{BIG}^{\text{ut}}$  follows Definition 11.2. We omit the type  $*$  and write the environments simply as sets of names.

**Definition 11.19** (encoding type derivations as bigraphs). *The encoding  $\llbracket - \rrbracket$  which takes an untyped  $\Lambda_{\text{sub}}$  derivation  $X \vdash t$  with  $\text{FV}(t) \subseteq X$  to a prime, ground bigraph  $g : \epsilon \rightarrow \langle 1, \mathcal{K}^*, X, *^X \rangle$  is defined inductively on the inference of  $X \vdash t$  and presented in Figure 11.7.*

The encoding maps a derivation of an untyped  $\Lambda_{\text{sub}}$  term to a prime bigraph with outer kind  $(1\text{var}, 1\text{lam}, 1\text{app}, 1\text{sub})^1$  and outer names of sort  $\nu$ . A quick check confirms that the encoding respects the sorting.

Any prime bigraph of  $'\Lambda\text{BIG}^{\text{ut}}$  with one exposed node which is not a **def** node corresponds to a  $\Lambda_{\text{sub}}$  term. We can show this by providing a translation from a subcategory  $'\Lambda\text{BIG}^{\text{ut}\lambda}$  of  $'\Lambda\text{BIG}^{\text{ut}}$  which is closed under reaction to  $\Lambda_{\text{sub}}$ .

The remaining terms of  $'\Lambda\text{BIG}^{\text{ut}}$  correspond to prime and parallel compositions of  $\Lambda_{\text{sub}}$  terms and bodies of substitution.

### 11.3.3 Dynamic properties

We have similar but stronger properties than  $'\Lambda\text{BIG}$ .

**Proposition 11.20** (reaction matches reduction).  $\llbracket X \vdash t \rrbracket \rightarrow g$  if and only if  $t \rightarrow_{\text{bcgc}} t'$  for some  $t'$  such that  $\llbracket X \vdash t' \rrbracket \simeq g$ .

By the encoding of  $\Lambda_{\text{sub}}$  into  $\Lambda\text{BIG}^{\text{ut}}$  and the translation mentioned above, the subcategory of prime bigraphs of  $\Lambda\text{BIG}^{\text{ut}}$  with kind sort  $(1\text{var}, 1\text{lam}, 1\text{app}, 1\text{sub})^1$  contains exactly the image of  $\Lambda_{\text{sub}}$  under the encoding. This subcategory is closed under reaction. By Proposition 11.20, Proposition 8.6, Lemma 8.7, and  $\rightarrow_{\text{cgC-UN}}$ , this subcategory encodes a conservative extension of the  $\lambda$ -calculus.

As with  $\Lambda\text{BIG}$ , we can transfer our proofs of confluence and normalisation from  $\Lambda_{\text{sub}}$  to the subcategory  $\Lambda\text{BIG}^{\text{ut}\lambda}$ .

### 11.3.4 Deficiency of the sorting

The sorting could be improved. Bigraphs whose outer interface does not contain the kind sort  $(1\text{def})^1$  are fine; we find these interesting as they model the concurrent computation of separate functional processes. However, problems arise when the outer interface contains the kind sort  $(1\text{def})^1$  which make it sometimes impossible to relate the bigraph to a tuple of  $\Lambda_{\text{sub}}$  terms.

Ideally, we would like to relate a ground bigraph  $g$  to a tuple of  $\Lambda_{\text{sub}}$  terms  $(t_1, \dots, t_n)$  such that if  $g \rightarrow_{\text{ACD}} g'$  then  $t_i \rightarrow_{\text{bcgc}} t'_i$  for some  $i \in [1, n]$ . However, how do we translate bigraphs with outer sort  $(1\text{def})^1$ ?

We propose three solutions for such a translation. The first simply discards the roots with sort  $(1\text{def})^1$ . However, this makes it impossible in general to match reaction with reduction as a  $\rightarrow_{\text{C}}$  reaction may involve an exposed **def** node. The second solution is to forget the exposed **def** node but translate its contents. Again, we cannot match reaction. The third solution is to: i) forget the **def** node (but keep its contents) if it does not link to any other nodes; and ii) introduce new closures in the translation otherwise so that *e.g.*  $\text{var}_x \parallel (\text{def}_x \circ \text{var}_y)$  translates to  $x[x/y]$ . Now reaction can be matched by reduction.

Almost. There are still some malformed bigraphs involving exposed **def** nodes. Consider  $g = \text{var}_z \parallel (\text{def}_{y:\delta} \circ (\text{var}_{x:\delta})) \parallel (\text{def}_{x:\delta} \circ (\text{var}_{y:\delta}))$ ,  $z = x$  or  $y$ . This bigraph may not be able to react<sup>6</sup> but it also breaks the structure of  $\Lambda_{\text{sub}}$ ; if an outer name  $x$  links a **def**-node and a **var**-node then we should consider the **def** node to be the body of substitution  $u$  and the **var**-node a free occurrence of  $x$  in  $t$  in the term  $t[x/u]$ . However, if we apply this intuition to the example above, we reach a paradox in  $g$  (try to create a valid  $\Lambda_{\text{sub}}$  encoding by enclosing the **def**-nodes in **sub**-nodes). Therefore we should disallow this situation in the sorting.

<sup>6</sup>We leave an explanation of why it cannot react as an exercise to the patient reader. The impatient reader is directed to Milner's explanation [107][p. 9].



A simpler solution could be to limit the number of roots which have an exposed **def**-node to one per interface. This breaks RPO creation but may still be sufficient for generating labels for the transition system. Otherwise, the sorting would have to disallow any **def** nodes or sites with sort  $(1\text{def})^1$  which are connected in a cycle (the example above is a two-cycle).

## 11.4 Modelling the simply typed $\lambda$ -calculus

In this section, we present a bigraphical model of simply typed  $\Lambda_{\text{sub}}$ . We begin by introducing simple types to  $\Lambda_{\text{sub}}$  calculus by using the typing rules of Di Cosmo and Kesner for  $\lambda\text{xgc}$  [51].

Again, the set of simple types over a countable set of ground types  $\mathcal{G}$  is given by the grammar:

$$\tau := \mathcal{G} \mid \tau \rightarrow \tau.$$

$G$  and  $G'$  denote ground types,  $A$ ,  $B$ , and  $C$  denote arbitrary types, and  $\alpha$  denotes function types  $A \rightarrow B$ . The typing rules of the simply typed  $\Lambda_{\text{sub}}$ -calculus are presented in Figure 11.8. Di Cosmo and Kesner previously used these to type  $\lambda\text{xgc}$  [51]. The rules extend the pure simple type system of Section 2.7 by adding a rule for explicit substitutions.

**Lemma 11.21** (Subject reduction). *If  $\Gamma \vdash t : A$  and  $t \rightarrow_{\text{bcgc}} u$  then  $\Gamma \vdash u : A$ .*

### 11.4.1 The sorting for $\Lambda\text{BIG}^{\rightarrow}$

Before we sort the signature, we extend it with ‘typed’ versions of the controls. The signature becomes an infinite set.

|                                                                                               |  |         |                                                                                         |  |        |
|-----------------------------------------------------------------------------------------------|--|---------|-----------------------------------------------------------------------------------------|--|--------|
| $\frac{}{\Gamma, x : A \vdash x : A}$                                                         |  | (axiom) | $\frac{\Gamma \vdash u : B \quad \Gamma, x : B \vdash t : A}{\Gamma \vdash t[x/u] : A}$ |  | (subs) |
| $\frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash (t u) : B}$ |  | (app)   | $\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \rightarrow B}$       |  | (abs)  |

**Figure 11.8:** Typing rules for simply typed  $\Lambda_{\text{sub}}$

**Definition 11.22** (The extended signature  $'\Lambda\text{BIG}^+$ ). *The set of controls  $\mathcal{K}^\rightarrow$  of  $'\Lambda\text{BIG}^+$  is*

$$\bigcup_{A,B \in \tau} \{\text{lam}^{A \rightarrow B} : 1, \text{app}^{A \rightarrow B, A} : 0, \text{var}^A : 1, \text{sub}^{B, A} : 1, \text{def}^A : 1, 1^A : 0, 2^A : 0, \text{U}^A : 0, \text{D}^A : 0\}.$$

*The controls  $\text{lam}^{A \rightarrow B}$  and  $\text{sub}^{B, A}$  are binding for all  $A, B$ . The other controls are nonbinding. The  $\text{var}^A$  controls are the only atomic controls. All other controls are active.*

The idea is to define one control per type so that the place graph structure reflects a type derivation. For example, the derivation  $y : B \vdash \lambda x. y : A \rightarrow B$  from  $x : A, y : B \vdash y : A$  is modelled by the bigraph  $(\text{lam}_{(x:A)}^{A \rightarrow B} \oplus \text{id}_{y:B}) \circ \text{var}_{y:B}^B$ . The next definition introduces an interpretation of types as sets of controls.

**Definition 11.23** (typed subsets of  $\mathcal{K}^\rightarrow$ ). *For any ground type  $G$  and function type  $\alpha$  we define the following:*

$$\begin{aligned} \mathcal{K}^G &= \{\text{var}^G\} \cup \bigcup_{B \in \tau} \{\text{app}^{B \rightarrow G, B}, \text{sub}^{G, B}\}, \\ \mathcal{K}^\alpha &= \{\text{var}^\alpha\} \cup \bigcup_{B \in \tau} \{\text{app}^{B \rightarrow \alpha, B}, \text{sub}^{\alpha, B}, \text{lam}^\alpha\}. \end{aligned}$$

**Notation.** *We use the following abbreviation for the kind function of a signature  $\mathcal{K}$  with maximum capacities. We write  $\text{kind}(K) = S$  where  $S \in \mathcal{P}(\mathcal{K}_{\text{vis}})$  to denote that  $\text{kind}(K)(K') = 1$  if  $K' \in S$  and  $\text{kind}(K)(K') = 0$  otherwise.*

**Definition 11.24** ( $'\Lambda\text{BIG}^\rightarrow$  signature). *The plain kind signature  $\mathcal{K}^\rightarrow$  with maximum capacities for  $'\Lambda\text{BIG}^\rightarrow$  extends the  $'\Lambda\text{BIG}^+$  signature with the kind function defined by*

$$\begin{aligned} \text{kind}(\text{lam}^{B \rightarrow A}) &= \text{kind}(\text{def}^A) = \text{kind}(1^A) = \text{kind}(2^A) = \text{kind}(\text{U}^A) &= \mathcal{K}^A \\ \text{kind}(\text{D}^A) &= \{\text{def}^A\} \\ \text{kind}(\text{var}^A) &= \text{kind}(\text{app}^{A \rightarrow B, A}) = \text{kind}(\text{sub}^{B, A}) &= \emptyset \\ \text{kind}_{\text{inv}}(\text{app}^{A \rightarrow B, A}) &= \{1^{A \rightarrow B}, 2^A\} \\ \text{kind}_{\text{inv}}(\text{sub}^{B, A}) &= \{\text{U}^B, \text{D}^A\} \end{aligned}$$

*for any  $A, B$ . The invisible controls are  $\bigcup_{A \in \tau} \{1^A, 2^A, \text{U}^A, \text{D}^A\}$ . The port of  $\text{lam}^{A \rightarrow B}$ ,  $\text{var}^A$ ,  $\text{sub}^{B, A}$ , and  $\text{def}^A$  controls has sort  $A$  for all  $A, B$ . The capacity of the sub and app multi-nodes is zero. All other non-atomic controls have a capacity of one.*

We depict the kind sorting with the schema in Figure 11.9, omitting capacities, where  $C$  is an arbitrary type and the dotted lines are filled when  $B$  is a function type.

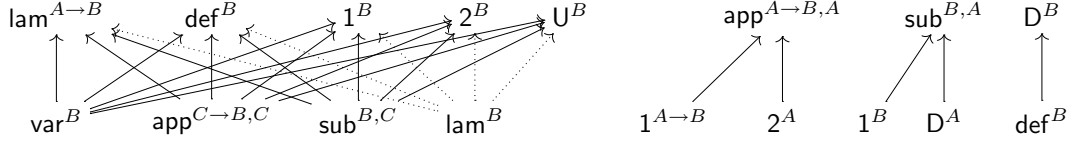


**Figure 11.9:** Representation of the simply typed  $'\Lambda\text{BIG}^{\rightarrow}$  sorting

As with  $'\Lambda\text{BIG}^{\text{ut}}$ , we will use a fitting (fully partitioned) subcategory but now our reasons are stronger. We wish for the encoding to respect the simply typed discipline of  $\Lambda_{\text{sub}}$  in some sense. To achieve this, the set of interface sorts should have some relationship to the set of types.

Ideally, we want a fitting s-category where the interface sorts are such that for any bigraph which correctly models a simply typed  $\Lambda_{\text{sub}}$  derivation:

1. there is exactly one choice of outer interface sort; and
2. the interface is related somehow to the derived type.

The first point implies that the interface sorts should be minimal in some sense and, with the second point, this suggests a solution. We will examine the reaction rules to demonstrate this as our set of interface sorts must be able to sort the reaction rules.

The reaction rules for  $'\Lambda\text{BIG}^{\rightarrow}$  are depicted in Figure 11.11. We have omitted the numbering of sites which is similar to Figure 11.6. This figure is a rule schema – for every pair  $(A, B)$  of types there are corresponding rules  $A^{A,B}$ ,  $C^A$ ,  $C^B$ , and  $D^{A,B}$ . If we forget the simple type annotations, they depict the rules of  $'\Lambda\text{BIG}^{\text{ut}}$ .

Consider the rule  $D^{A,B}$ . The redex must have a sort containing  $\text{sub}^{B,A}$ . The site which persists in the reactum has kind sort  $\mathcal{K}^B$  which includes  $\text{sub}^{B,A}$ . Therefore,  $\mathcal{K}^B$  seems to be an appropriate outer sort for the rule. A  $D^{A,B}$  reaction may sometimes follow a  $A^{A,B}$  reaction so  $\mathcal{K}^B$  also seems an appropriate sort for  $A^{A,B}$ .

This suggests a sorting – we base the allowed sorts for interfaces on the set of simple types and use a fully partitioned s-category, keeping  $\text{def}$ -nodes separate. We define the set of interface sorts as follows. Each type  $A \in \tau$  is modelled with two interface sorts –  $\mathcal{K}^A$  (used for all  $\Lambda_{\text{sub}}$  terms) and  $\{\text{def}^A\}$  (used for bodies of substitution). The reaction rules have the only possible outer sorts.

**Definition 11.25** ( $'\Lambda\text{BIG}^{\text{ks}}$ ). *The Brs  $'\Lambda\text{BIG}^{\text{ks}}$  is the reactive system over the fully partitioned subcategory of  $\text{BIG}_h(\Sigma_K)$ , the kind sorted Brs over  $\mathcal{K}^{\rightarrow}$  with semi-rigid maximum capacities,*

where the set of interface kinds is generated from<sup>7</sup> the pair  $\{\text{var}, \text{lam}, \text{app}, \text{sub}\}$  and  $\{\text{def}\}$  and the reaction rules are as depicted in Figure 11.11.

**Definition 11.26** ( $\Lambda\text{BIG}^{\text{ks}_1}$ ). *The Brs  $\Lambda\text{BIG}^{\text{ks}_1}$  is defined by taking the largest full subcategory of  $\Lambda\text{BIG}^{\text{ks}}$  where if  $\text{kind}(r)(\text{def}^A) > 0$  then  $\text{kind}(r)(\text{def}^A) = 1$  and where capacities of places are not zero or  $\odot$ .*

**Definition 11.27** ( $\Lambda\text{BIG}^{\text{s}}$ ). *The Brs  $\Lambda\text{BIG}^{\text{s}}$  is defined by applying the  $\sigma\lambda\nu\delta$ -sorting to  $\Lambda\text{BIG}^{\text{ks}_1}$ .*

**Definition 11.28** ( $\Lambda\text{BIG}^{\text{ps}}$ ). *The Brs  $\Lambda\text{BIG}^{\text{ps}}$  is defined by applying a plain sorting to  $\Lambda\text{BIG}^{\text{s}}$  where the set of name sorts is the set of simple types.*

**Definition 11.29** ( $\Lambda\text{BIG}^{\rightarrow}$ ). *The Brs  $\Lambda\text{BIG}^{\rightarrow} = \Lambda\text{BIG}^{\text{s}} \times_{\Lambda\text{BIG}^+} \Lambda\text{BIG}^{\text{ps}}$  is defined as the paired sorting of  $\Lambda\text{BIG}^{\text{s}}$  and  $\Lambda\text{BIG}^{\text{ps}}$ . The ions and reaction rules of  $\Lambda\text{BIG}^{\rightarrow}$  are sorted as depicted in Figures 11.10 and 11.11 respectively<sup>8</sup>.*

When a bigraph  $G$  of  $\Lambda\text{BIG}^{\text{ks}}$  is prime and the root has one child, we write  $\text{sort}(G)$  for the kind sort of the root, omitting capacity.

**Proposition 11.30.**  $\mathcal{U}^{\text{ks}_1}$  creates RPOs and reflects pushouts.

*Proof.* Follows similarly to Proposition 11.16. □

**Proposition 11.31.** The functor  $\mathcal{U}^{\text{ks}} \circ \mathcal{U}^{\text{ks}_1} \circ \mathcal{U}^{\text{s}} \circ \pi_2$  creates RPOs and reflects pushouts.

*Proof.* Follows similarly to Proposition 11.17. □

**Corollary 11.32.** Wide bisimilarity over the standard transition system ST for  $\Lambda\text{BIG}^{\rightarrow}$  is a congruence and mono bisimilarity is a congruence for mono contexts.

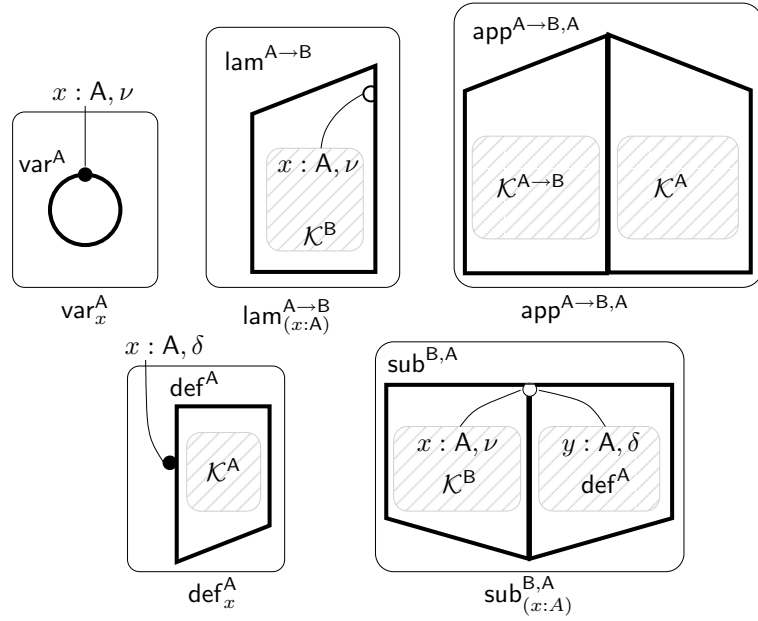
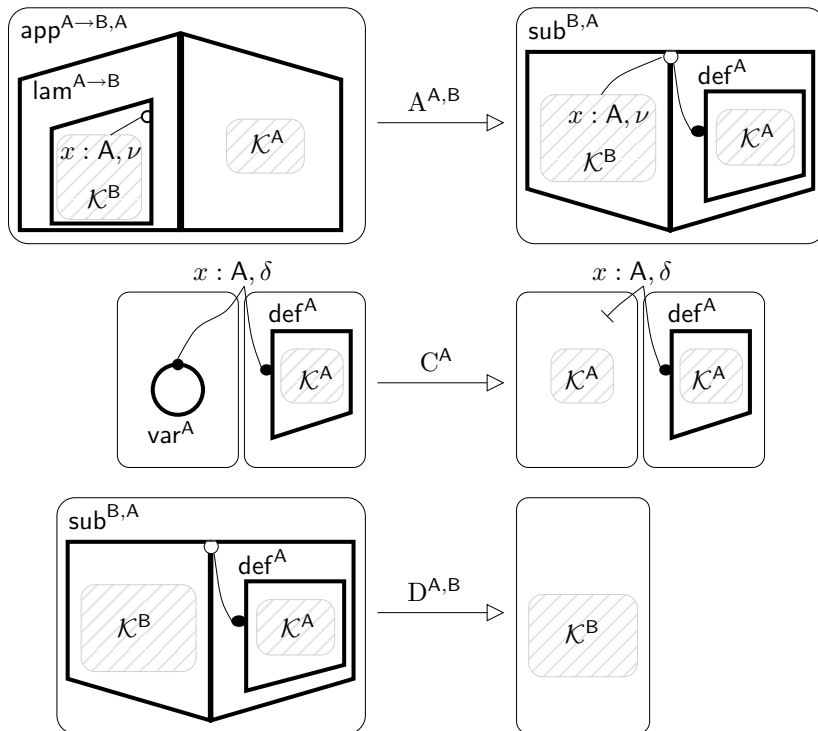
### 11.4.2 Static correspondence

We now encode simply-typed derivations of  $\Lambda_{\text{sub}}$  terms in  $\Lambda\text{BIG}^{\rightarrow}$ . The use of a partitioned subcategory allows the sort of an encoding of a term  $t$  to match the derived type of  $t$  under some environment.

The encoding of a derivation  $\Gamma \vdash t : A$  is indexed with an environment  $\Gamma'$  where  $\Gamma \subseteq \Gamma'$  and  $\text{BV}(t) \cap \text{dom}(\Gamma') = \emptyset$ . This indexing is required in order to prove that  $\Lambda_{\text{sub}}$  reduction matches  $\Lambda\text{BIG}^{\rightarrow}$  reaction as the former loses free variables whereas the latter does not. We let environments  $\Gamma$  represent a plain sorting of  $\text{dom}(\Gamma)$  to ease the encoding.

<sup>7</sup>For convenience, we omit capacities when specifying the set of interface sorts (see Definition 4.24).

<sup>8</sup>We omit the  $\sigma\lambda\nu\delta$ -sorts in the figures for clarity. See Figure 11.11 for the appropriate types.

Figure 11.10: Ion schema for  $\lambda\text{BIG}^{\rightarrow}$ Figure 11.11: Parametric reaction rule schema for  $\lambda\text{BIG}^{\rightarrow}$

| Derivation                                    | Encoding with index $\Gamma'$ (all variables in $\Gamma'$ have $\sigma\lambda\nu\delta$ -sort $\nu$ )                                                                                                                                                                                                                                                             |
|-----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\Gamma, x : A \vdash x : A$                  | $\text{var}_{x:A,\nu}^A \oplus \Gamma' / \{x : A, \nu\}$                                                                                                                                                                                                                                                                                                          |
| $\Gamma \vdash \lambda x.t : A \rightarrow B$ | $(\text{lam}_{(x:A,\lambda)}^{A \rightarrow B} \oplus \text{id}_{\Gamma'}) \llbracket \Gamma, x : A, \nu \vdash t : B \rrbracket_{\Gamma', x:A,\nu}$                                                                                                                                                                                                              |
| $\Gamma \vdash t u : A$                       | $(\text{app}^{B \rightarrow A, B} \oplus (\text{id}_{\Gamma'} \mid \text{id}_{\Gamma'})) \circ (\llbracket \Gamma \vdash t : B \rightarrow A \rrbracket_{\Gamma'} \parallel \llbracket \Gamma \vdash u : B \rrbracket_{\Gamma'})$                                                                                                                                 |
| $\Gamma \vdash t[x/u] : A$                    | $(\text{sub}_{(x:B,\delta)}^{A,B} \oplus (\text{id}_{\Gamma'} \mid \text{id}_{\Gamma'})) \circ \{x : B, \delta/x : B, \nu \bar{y} : B, \delta\}$<br>$\circ (\llbracket \Gamma, x : B, \nu \vdash t : A \rrbracket_{\Gamma', x:B,\nu} \parallel (\text{def}_{\bar{y}:B,\delta}^B \oplus \text{id}_{\Gamma'}) \llbracket \Gamma \vdash u : B \rrbracket_{\Gamma'})$ |

**Figure 11.12:** Encoding of simply typed  $\Lambda_{\text{sub}}$  terms into  $'\Lambda\text{BIG}^{\rightarrow}$ 

**Definition 11.33** (encoding type derivations as bigraphs). *The encoding  $\llbracket - \rrbracket_{\Gamma'}$  which takes a simply typed  $\Lambda_{\text{sub}}$  derivation  $\Gamma \vdash t : A$  with  $\Gamma \subseteq \Gamma'$  to a prime, ground bigraph  $g : \epsilon \rightarrow \langle 1, \mathcal{K}^A, \text{dom}(\Gamma'), \Gamma' \rangle$  is defined inductively on the inference of  $\Gamma \vdash t : A$  and presented in Figure 11.12.*

The encoding preserves types in some way; if  $\Gamma \vdash t : A$  then  $\text{sort}[\llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'}] = \mathcal{K}^A$ .

**Proposition 11.34.** *The encoding  $\llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'}$  of a simply typed  $\Lambda_{\text{sub}}$  derivation is well-sorted with outer kind sort  $\mathcal{K}^A$  and link sort  $\Gamma'$ .*

*Proof.* See Appendix C.4.1, Proposition C.8. □

**Proposition 11.35.** *If  $t \equiv_{\alpha} u$ ,  $\Gamma \vdash t : A$ , and  $\Gamma \subseteq \Gamma'$  then  $\llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'} \simeq \llbracket \Gamma \vdash u : A \rrbracket_{\Gamma'}$ .*

*Proof.* Induct over the structure of  $t$  where  $\lambda x.t \equiv_{\alpha} \lambda y.t\{y/x\}$ ,  $y \notin \text{FV}(t)$  and  $t[x/u] \equiv_{\alpha} t\{y/x\}[y/u]$ ,  $y \notin \text{FV}(t)$  are the non-trivial cases. □

### 11.4.3 Dynamic properties

We have the following results which are similar to  $'\Lambda\text{BIG}$  except that now each of the three rules for  $\Lambda_{\text{sub}}$  is matched by an infinite number of  $'\Lambda\text{BIG}^{\rightarrow}$  rules<sup>9</sup>.

**Proposition 11.36** (reaction matches reduction).  *$\llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'} \rightarrow g$  if and only if  $t \rightarrow_{\text{bcgc}} t'$  for some  $t'$  such that  $\llbracket \Delta \vdash t' : A \rrbracket_{\Gamma'} \simeq g$  where  $\Delta \subseteq \Gamma$ .*

<sup>9</sup>The number of rules is infinite, but if e.g. a  $\rightarrow_c$  redex exists at some place in a term then only one typed  $\rightarrow_c$  rule can apply at that place in the encoding.

$'\Lambda\text{BIG}^\rightarrow$  was built by further sorting  $'\Lambda\text{BIG}^{\text{ut}}$ . We can introduce a forgetful (but not faithful) functor between the two systems.

**Definition 11.37** ( $\mathcal{U}^\rightarrow$ ). *We define the functor  $\mathcal{U}^\rightarrow : '\Lambda\text{BIG}^\rightarrow \rightarrow '\Lambda\text{BIG}^{\text{ut}}$  as follows. Let the mapping  $u : \mathcal{K}^\rightarrow \rightarrow \mathcal{K}^{\text{ut}}$  be defined as*

$$\mathcal{K}^A \mapsto \mathcal{K}^*, \{\text{def}^A\} \mapsto \{\text{def}\} \text{ for all } A.$$

*On objects,  $\mathcal{U}^\rightarrow(I) = \mathcal{U}(I)$  is the untyped interface of  $I$ , where kind sorts of  $I$  are mapped to their  $u$ -images and name types are mapped to the type  $*$ . On ground arrows,  $\mathcal{U}^\rightarrow(G)$  is defined by  $\text{ctrl}_{\mathcal{U}^\rightarrow(G)}(v) = u(\text{ctrl}_G(v))$ , retaining the link and prnt functions.  $\mathcal{U}^\rightarrow(G)$  is called the untyped bigraph of  $G$ . Given a ground reaction rule of  $'\Lambda\text{BIG}^\rightarrow$ , the underlying ground reaction rule is defined as the pair of untyped redex and untyped reactum. The underlying ground reaction rules are exactly the reaction rules of  $'\Lambda\text{BIG}^{\text{ut}}$ .*

We can compose  $\mathcal{U}^\rightarrow$  with the encoding  $\llbracket \cdot \rrbracket_{\Gamma'}$  of simply typed  $\Lambda_{\text{sub}}$  derivations to retrieve the encoding  $\llbracket \cdot \rrbracket_{\text{dom}(\Gamma')}$  of  $\Lambda_{\text{sub}}$  terms into  $'\Lambda\text{BIG}^{\text{ut}}$ .

**Proposition 11.38.**  $\mathcal{U}^\rightarrow \circ \llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'} = \llbracket X \vdash t : * \rrbracket$  where  $\text{dom}(\Gamma') = X$ .

*Proof.* Induction over the structure of  $t$ . The cases are sketched below.

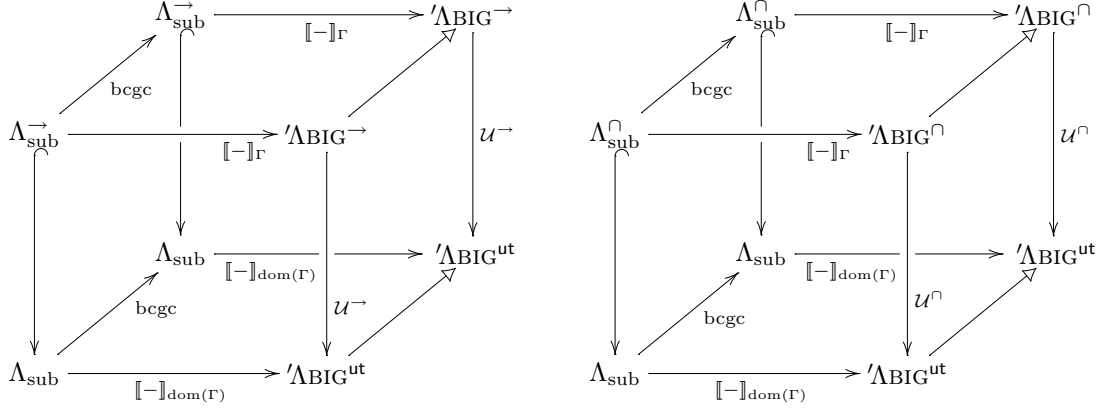
$$\begin{aligned} \mathcal{U}^\rightarrow \circ \llbracket \Gamma, x : A \vdash x : A \rrbracket_{\Gamma'} &= \text{var}_{x:\nu} \oplus X / \{x : \nu\} \\ \mathcal{U}^\rightarrow \circ \llbracket \Gamma \vdash \lambda x. t : A \rightarrow B \rrbracket_{\Gamma'} &= (\text{lam}_{(x:\lambda)} \oplus \text{id}_X) \circ (\mathcal{U}^\rightarrow \circ \llbracket \Gamma, x : A, \nu \vdash t : B \rrbracket_{\Gamma', x:A, \nu}) \\ \mathcal{U}^\rightarrow \circ \llbracket \Gamma \vdash t \ u : A \rrbracket_{\Gamma'} &= (\text{app} \oplus (\text{id}_X \mid \text{id}_X)) \\ &\quad \circ (\mathcal{U}^\rightarrow \circ \llbracket \Gamma \vdash t : B \rightarrow A \rrbracket_{\Gamma'} \parallel \mathcal{U}^\rightarrow \circ \llbracket \Gamma \vdash u : B \rrbracket_{\Gamma'}) \\ \mathcal{U}^\rightarrow \circ \llbracket \Gamma \vdash t[x/u] : A \rrbracket_{\Gamma'} &= (\text{sub}_{(x:\delta)} \oplus (\text{id}_X \mid \text{id}_X)) \circ \{x : \delta/x : \nu \ \bar{y} : \delta\} \\ &\quad \circ (\mathcal{U}^\rightarrow \circ \llbracket \Gamma, x : B, \nu \vdash t : A \rrbracket_{\Gamma', x:B, \nu} \\ &\quad \parallel (\text{def}_{\bar{y}:\delta} \oplus \text{id}_X)(\mathcal{U}^\rightarrow \circ \llbracket \Gamma \vdash u : B \rrbracket_{\Gamma'})) \end{aligned}$$

□

For convenience, we denote the mapping with environment  $\Gamma$  from  $\Lambda_{\text{sub}}$  terms to  $'\Lambda\text{BIG}^{\text{ut}}$  agents as  $\llbracket \cdot \rrbracket_{\text{dom}(\Gamma)}$  in Propositions 11.39 and 11.70 and in Figure 11.13.

**Proposition 11.39.**  *$'\Lambda\text{BIG}^{\text{ut}}$  reaction on  $\llbracket \cdot \rrbracket_{\text{dom}(\Gamma)}$ -images strongly simulates  $'\Lambda\text{BIG}^\rightarrow$  reaction on  $\llbracket \cdot \rrbracket_\Gamma$ -images along  $\mathcal{U}^\rightarrow$ .*

*Proof.* The proof follows by the diagram on the left of Figure 11.13 where  $\Lambda_{\text{sub}}^\rightarrow$  denotes the subcalculus of  $\Lambda_{\text{sub}}$ , closed under reduction, of terms typable under the simply typed discipline. Proposition

**Figure 11.13:** Reduction cubes for  $'\Lambda\text{BIG}^{\rightarrow}$  and  $'\Lambda\text{BIG}^{\cap}$ 

11.38 proves the squares, Proposition 11.36 proves the ‘top face’ of the cube, and the matching of  $\Lambda_{\text{sub}}$  reduction by  $'\Lambda\text{BIG}^{\text{ut}}$  reaction (Proposition 11.20) proves the ‘bottom face.’  $\square$

This enables us to reflect some properties of  $'\Lambda\text{BIG}^{\text{ut}}$  back through  $\mathcal{U}^{\rightarrow}$ . Specifically, we can transfer our proofs of confluence and normalisation from  $\Lambda_{\text{sub}}$  to the subcategory of prime, ground bigraphs of  $'\Lambda\text{BIG}^{\rightarrow}$  with outer kind sorts  $1 \times \mathcal{K}^A$  for all  $A \in \tau$ .

**Proposition 11.40.**  *$'\Lambda\text{BIG}^{\rightarrow}$  can simulate  $\beta$ -reduction, is confluent on terms without metavariables, preserves strong normalisation of  $\beta$ -reduction, and has full composition of substitutions on encodings of simply typed  $\Lambda_{\text{sub}}$  derivations.*

**Proposition 11.41.** *Simply typed terms of  $\Lambda_{\text{sub}}$  are strongly normalising.*

*Proof.* We use Herbelin’s approach [69] and the translation  $\mathbb{C}$  of Definition 9.29. We have  $\mathbb{C}(t) \rightarrow_b t$  for all  $t$ . By inducting over the number of explicit substitutions in  $t$ , and using the fact that  $\rightarrow_b$  preserves types, we can prove that if  $\Gamma \vdash t : A$  then  $\Gamma \vdash \mathbb{C}(t) : A$ . Moreover, the derivation of  $\Gamma \vdash \mathbb{C}(t) : A$  uses the rules of the simply typed  $\lambda$ -calculus. Therefore,  $\mathbb{C}(t)$  is strongly normalising for  $\beta$ -reduction [8]. As  $\Lambda_{\text{sub}}$  has PSN, we conclude that  $t$  is strongly normalising.  $\square$

We have the following corollary of Propositions 11.36 and 11.41.

**Corollary 11.42.**  *$'\Lambda\text{BIG}^{\rightarrow}$  is strongly normalising on encodings of simply typed  $\Lambda_{\text{sub}}$  derivations.*

**Corollary 11.43.** *Ground bigraphs of  $'\Lambda\text{BIG}^{\rightarrow}$  whose outer interface does not contain a kind sort  $(1\text{def}^A)^1$  for any  $A \in \tau$  are strongly normalising.*



*Proof.* Such bigraphs can be written as prime and parallel products of  $\llbracket - \rrbracket_\Gamma$ -images of simply typed  $\Lambda_{\text{sub}}$  derivations and cannot react together.  $\square$

It is likely that all bigraphs in  $\Lambda\text{BIG}^\rightarrow$  are strongly normalising but due to the current deficiency of the sorting (see Section 11.3.4), we cannot immediately conclude this.

## 11.5 Modelling the $\lambda$ -calculus with intersection types

Our final model of  $\lambda$ -calculus improves on  $\Lambda\text{BIG}^\rightarrow$  by considering intersection types. Intersection types provide a characterisation of the set of strongly normalising  $\lambda$ -terms and as we sketched in Section 9.4, they also characterise the set of strongly normalising  $\Lambda_{\text{sub}}$ -terms. Our goal is to define a Brs which models exactly the set of terminating functions (and parallel compositions thereof). We show here that we can model at least that set but we still require to solve the deficiency of Section 11.3.4.

We use the additive intersection type system of Section 9.4 (see Figure 9.5) on  $\Lambda_{\text{sub}}$ . Again, the set of intersection types over a set of ground types  $\mathcal{G}$  is given by the grammar:

$$\tau ::= G \mid \tau \rightarrow \tau \mid \tau \wedge \tau \text{ where } G \in \mathcal{G}.$$

### 11.5.1 Bounded completeness of the type preorder

We will use a subcategory of a kind sorted s-category to model  $\Lambda\text{BIG}^\cap$ , our model of  $\lambda$ -calculus with intersection types. In order to prove RPO creation, it suffices that the subcategory has the property of bounded completeness. The set of interface sorts for the subcategory will be a semantic interpretation  $\mathcal{K}$  of the set of intersection types as sets of controls with the property that if  $A \ll B$  then  $\mathcal{K}^A \subseteq \mathcal{K}^B$ . Therefore, we require that the set of intersection types is bounded complete (up to equivalence) with respect to  $\ll$ .

Recall that the preorder  $\ll$  is given in Definition 2.56. A standard equivalence of intersection types is as follows.

**Definition 11.44.**  $A \sim B$  iff  $A \ll B$  and  $B \ll A$ .

**Lemma 11.45.**

1.  $A \wedge B \sim B \wedge A$ .
2.  $(A \wedge B) \wedge C \sim A \wedge (B \wedge C)$ .

3.  $A \wedge A \sim A$ .

**Corollary 11.46.**

1.  $A \ll B \wedge D \text{ iff } A \ll D \wedge B$
2.  $A \ll (B \wedge C) \wedge D \text{ iff } A \ll B \wedge (C \wedge D)$

As  $\wedge$  is commutative and associative up to  $\sim$ , we let  $\wedge_{\underline{n}} A_i$  denote  $A_1 \wedge \dots \wedge A_n$ .

A proof of bounded completeness for the  $\ll$  relation has not been previously published to our knowledge. We suspect that the reason for this is that it is not typically of interest as it is related more to union types rather than intersection types.

**Proposition 11.47** ( $\ll_{/\sim}$  is bounded complete). *If  $A_i \ll C$  for all  $i \in \underline{n}$  then there exists a type  $\vee_{\underline{n}} A_i$  such that for all  $D$  where  $A_i \ll D$  for all  $i \in \underline{n}$  we have  $\vee_{\underline{n}} A_i \ll D$ .*

*Proof.* See Appendix C.4.2, Proposition C.12. □

Besides the intersection of types (and hence of interface sorts), we also consider the *union*, or *join*, of types, to define the RPO interface. The function  $P : \tau \rightarrow \mathcal{P}(\tau)$  in the following takes a type to its set of basic types (see Definition C.9 on page 84).

**Definition 11.48** (partial join (up to  $\sim$ )). *The join  $\vee_{\underline{n}} A_i$  of a set of types is defined when  $\cap_{\underline{n}} P(A_i) \neq \emptyset$  as  $\vee_{\underline{n}} A_i = \wedge(\cap_{\underline{n}} P(A_i))$ .*

**Lemma 11.49.** *When  $\vee_{\underline{n}} A_i$  is defined then  $A_j \ll \vee_{\underline{n}} A_i$  for all  $j \in \underline{n}$ .*

*Proof.* Follows by Lemma C.11, page 84, Appendix C.4.2. □

The join construction will be used in a proof of relative pushout (RPO) creation for  $\lambda\text{BIG}^\square$ . The property of an s-category with RPOs is somewhat analogous to the property of a partial order being bounded complete and is particularly close in our model. For example, the diagram on the left of Figure 11.14 depicts an RPO. Given a bound  $G_0, G_1$  for a pair of arrows  $F_0, F_1$ , an RPO is a least bound  $H_0, H_1$  with an mediating arrow  $H$  to the larger bound. The diagram in the middle depicts a join  $A \vee B$  of two types  $A$  and  $B$ . Given a bound  $C$  for a pair of types  $A$  and  $B$ , a join is a least bound  $A \vee B$  such that  $A \vee B \ll C$ . The analogy will hopefully become clearer when we present our model and the proof of RPO creation. In our model, we use a set-theoretic interpretation of types as sets of controls where  $\ll$  is modelled by subset inclusion,  $\wedge$  by set intersection, and  $\vee$  somewhat by set union. The proof of RPO creation boils down to a proof that given a set

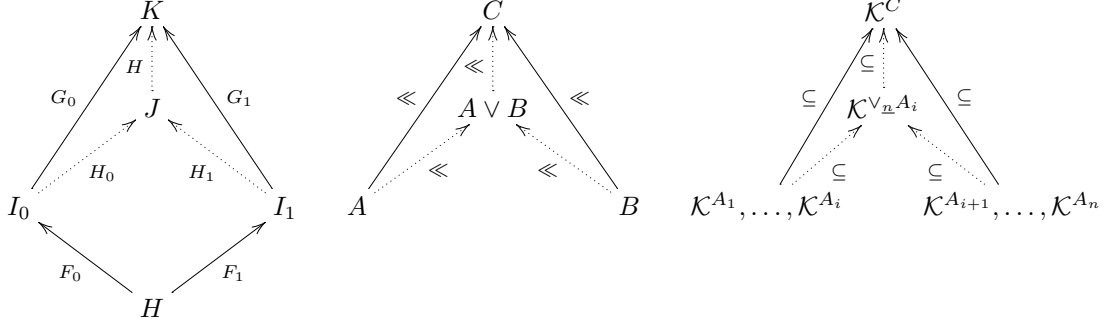


**Figure 11.14:** An RPO, a join in  $(\tau, \ll)$ , and a join in  $(\mathcal{P}(\mathcal{K}), \subseteq)$ .

of type interpretations  $\mathcal{K}^{A_i}$ ,  $i \in \underline{n}$ , their union is a valid type interpretation *i.e.* that the diagram on the right of the figure is valid and depicts a join. Proposition 11.47 ensures the validity and union forms joins over  $(\mathcal{P}(\mathcal{K}), \subseteq)$ .

### 11.5.2 The sorting for $\lambda\text{BIG}^\cap$

We extend the  $\lambda\text{BIG}$  signature with intersection-typed versions of the controls.

**Definition 11.50** (The extended signature  $\lambda\text{BIG}^+$ ). *The set of controls  $\mathcal{K}^\wedge$  of  $\lambda\text{BIG}^+$  is defined as:*

$$\begin{aligned} \mathcal{K}^\wedge = & \bigcup_{A_1, \dots, A_n, B_1, \dots, B_n \in \tau} \{ \text{lam}^{\wedge_{\underline{n}}(A_i \rightarrow B_i)} : 1, \text{app}^{\wedge_{\underline{n}}(A_i \rightarrow B_i), \wedge_{\underline{n}} A_i} : 0 \} \\ & \cup \bigcup_{A, B \in \tau} \{ \text{var}^A : 1, \text{sub}^{B, A} : 1, \text{def}^A : 1, 1^A : 0, 2^A : 0, \text{U}^A : 0, \text{D}^A : 0 \}. \end{aligned}$$

Controls  $\text{lam}^{\wedge_{\underline{n}}(A_i \rightarrow B_i)}$  and  $\text{sub}^{B, A}$  are binding with arity 1.  $\text{var}^A$  and  $\text{def}^A$  controls have arity 1 and are nonbinding. All other controls are nonbinding with arity 0. The  $\text{var}^A$  controls are the only atomic controls. All other controls are active.

Again we have one control per type to allow the place graph structure to reflect type derivations. However, our interpretation of types as sets as controls now considers the subtyping relation  $\ll$  and maps a type  $A$  to the set of all controls whose ‘type’ is a subtype of  $A$ .

**Definition 11.51** (main control sorts). *The interpretation of a type  $A$  as a set of controls is given*

by:

$$\begin{aligned} \mathcal{K}^A = & \bigcup_{B \ll A} (\{\text{var}^B\} \cup \bigcup_{C \in \tau} \{\text{sub}^{B,C}\}) \\ & \cup \bigcup_{\wedge_{\underline{n}} B_i \ll A, C_1, \dots, C_n \in \tau} \{\text{app}^{\wedge_{\underline{n}}(C_i \rightarrow B_i), \wedge_{\underline{n}} C_i}\} \\ & \cup \bigcup_{\wedge_{\underline{n}} (B_i \rightarrow A_i) \ll A} \{\text{lam}^{\wedge_{\underline{n}}(B_i \rightarrow A_i)}\}. \end{aligned}$$

**Definition 11.52** ( $(\mathcal{K}^{\wedge\lambda}, \text{Int}(\mathcal{K}^{\wedge}))$ ). We define the set  $\mathcal{K}^{\wedge\lambda} = \{\mathcal{K}^A \mid A \in \tau\}$  and the set  $\text{Int}(\mathcal{K}^{\wedge}) = \mathcal{K}^{\wedge\lambda} \cup \{\{\text{def}^A\} \mid A \in \tau\}$ .

We will take  $\text{Int}(\mathcal{K}^{\wedge})$  as our set of interface sorts. We require certain properties of the partial order  $(\text{Int}(\mathcal{K}^{\wedge}), \subseteq)$  for RPO creation and pushout reflection for jointly opcartesian bounds. Note that for any type  $A$ ,  $\text{def}^A$  is not comparable with any other element of the order besides itself.

**Lemma 11.53** (properties of the interpretation).

1.  $A \ll B$  iff  $\mathcal{K}^A \subseteq \mathcal{K}^B$ ;
2. If  $A \sim B$  then  $\mathcal{K}^A = \mathcal{K}^B$ ;
3. If  $A \ll B$  then  $\mathcal{K}^A = \mathcal{K}^{A \wedge B}$ ;
4.  $\mathcal{K}^{A \wedge B} = \mathcal{K}^{B \wedge A}$ ;
5.  $\mathcal{K}^{(A \wedge B) \wedge C} = \mathcal{K}^{A \wedge (B \wedge C)}$ ;
6.  $\mathcal{K}^{A \wedge B} = \mathcal{K}^A \cap \mathcal{K}^B$ .
7. If  $A \vee B$  is defined then  $\mathcal{K}^A \subseteq \mathcal{K}^{A \vee B}$ .
8. If  $A \vee B$  is defined then  $\mathcal{K}^{A \vee B}$  is the join of  $\mathcal{K}^A$  and  $\mathcal{K}^B$  in  $(\text{Int}(\mathcal{K}^{\wedge}), \subseteq)$ .

*Proof.* 1 follows by Definition 11.51. 2 follows by definition of  $\sim$  and 1. For 3, we show that if  $A \ll B$  then  $A \sim A \wedge B$  and then use 2. 4 and 5 follow from 2 and Lemma 11.46. For 6, use Definition 2.56.5. For 7, use 1 and Lemma 11.49. 8 follows by 1 and the fact that  $A \vee B$  is a join in  $(\tau, \ll)$ .  $\square$

Note that  $\mathcal{K}^{A_1} \cup \mathcal{K}^{A_2} \subset \mathcal{K}^{A_1 \vee A_2}$  in general and so  $\mathcal{K}^{A_1 \vee A_2}$  is not the join of  $\mathcal{K}^{A_1}$  and  $\mathcal{K}^{A_2}$  in  $(\mathcal{P}(\mathcal{K}^{\wedge}), \subseteq)$ . However,  $\mathcal{K}^{A_1 \vee A_2}$  is their join in  $(\mathcal{K}^{\wedge\lambda}, \subseteq)$  which is sufficient for a proof of RPO creation and also motivates restricting the interfaces in  $'\Lambda \text{BIG}^{\cap}$ .

**Definition 11.54** ( $(\llbracket - \rrbracket_{\tau})$ ). The function  $\llbracket - \rrbracket_{\tau} : (\tau, \ll) \rightarrow (\text{Int}(\mathcal{K}^{\wedge}), \subseteq)$  between the preorders on types and their interpretation is defined as  $\llbracket A \rrbracket_{\tau} = \mathcal{K}^A$ . By Lemma 11.53, we observe that the function preserves order, meets, and joins (when the last exist).

We consider the set  $\mathcal{K}^B$  to contain controls which can resolve to type  $B$ . Lemma 11.53.6 describes how the intention of the  $\wedge \text{E}$  and  $\wedge \text{I}$  rules of the typing system are captured somewhat by Definition 11.51;  $K \in \mathcal{K}^{A \wedge B}$  iff  $K \in \mathcal{K}^A$  and  $K \in \mathcal{K}^B$ .

**Definition** ( $'\Lambda\text{BIG}^\cap$  signature). *The plain kind signature  $\mathcal{K}^\wedge$  with maximum capacities for  $'\Lambda\text{BIG}^\cap$  extends the  $'\Lambda\text{BIG}^+$  signature with the kind function defined by*

$$\begin{aligned} \text{kind}(\text{lam}^{\wedge_{\underline{n}}(B_i \rightarrow A_i)}) &= \mathcal{K}^{\wedge_{\underline{n}} A_i} \\ \text{kind}(\text{def}^A) = \text{kind}(1^A) = \text{kind}(2^A) = \text{kind}(\text{U}^A) &= \mathcal{K}^A \\ \text{kind}(\text{D}^A) &= \{\text{def}^A\} \\ \text{kind}(\text{var}^A) = \text{kind}(\text{app}^{A \rightarrow B, A}) = \text{kind}(\text{sub}^{B, A}) &= \emptyset \\ \text{kind}_{\text{inv}}(\text{app}^{\wedge_{\underline{n}}(A_i \rightarrow B_i), \wedge_{\underline{n}} A_i}) &= \{1^{\wedge_{\underline{n}}(A_i \rightarrow B_i)}, 2^{\wedge_{\underline{n}} A_i}\} \\ \text{kind}_{\text{inv}}(\text{sub}^{B, A}) &= \{\text{U}^B, \text{D}^A\} \end{aligned}$$

for any  $A, B, A_i, B_i, i \in \underline{n}$ . The invisible controls are  $\bigcup_{A \in \tau} \{1^A, 2^A, \text{U}^A, \text{D}^A\}$ .  $\text{lam}^{\wedge_{\underline{n}}(A_i \rightarrow B_i)}$  controls have link sort  $\wedge_{\underline{n}} A_i$ , for all  $A_i, B_i$ .  $\text{var}^A$ ,  $\text{sub}^{B, A}$ , and  $\text{def}^A$  controls have link sort  $A$  for all  $B$ . The capacity of the **sub** and **app** multi-nodes is zero. All other non-atomic controls have a capacity of one.

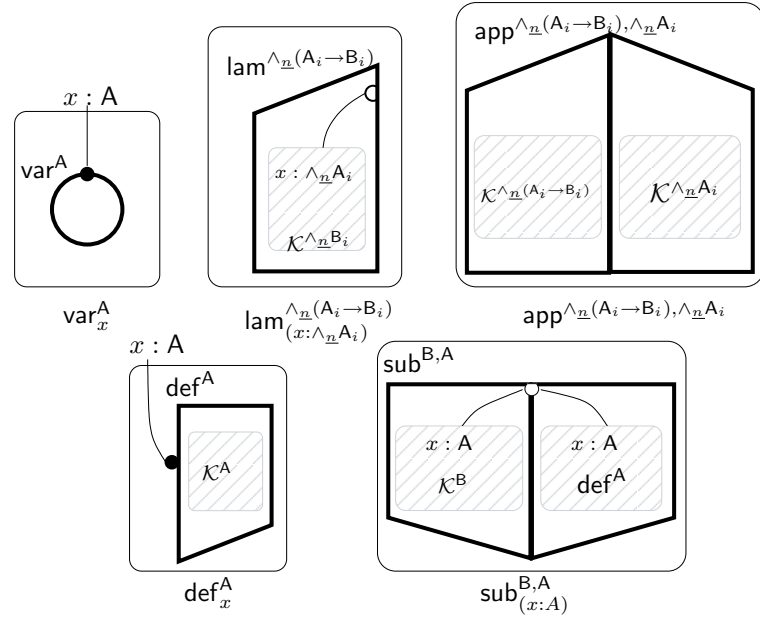
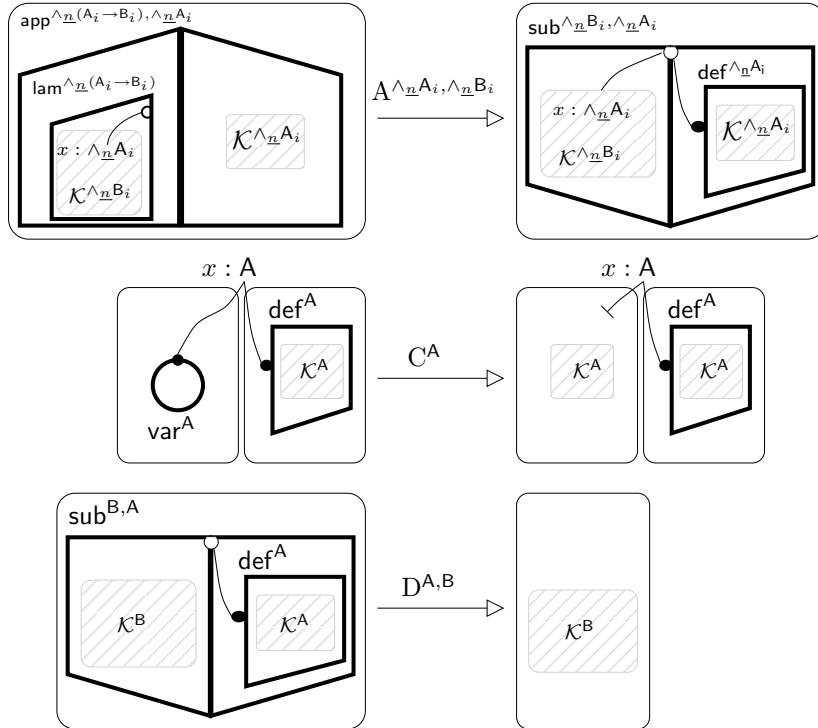
The main controls (as ions) are depicted in Figure 11.15. The sites in Figure 11.15 are labelled with their interface sort. This implies that *e.g.* a  $\text{lam}^{\wedge_{\underline{n}}(B_i \rightarrow A_i)}$  node can have a child whose control is in  $\mathcal{K}^{\wedge_{\underline{n}} B_i}$ . The outer sorts of the ions in the figure are, in order of presentation,  $\mathcal{K}^A$ ,  $\mathcal{K}^{\wedge_{\underline{n}}(A_i \rightarrow B_i)}$ ,  $\mathcal{K}^{\wedge_{\underline{n}} B_i}$ ,  $\mathcal{K}^A$ , and  $\mathcal{K}^B$ . These are the smallest sorts which can sort the respective ions.

We chose the model to respect the intersection typing rules for  $\Lambda_{\text{sub}}$ . For example, an application node  $\text{app}^{\wedge_{\underline{n}}(A_i \rightarrow B_i), \wedge_{\underline{n}} A_i}$  node can contain something of sort  $\mathcal{K}^{\wedge_{\underline{n}}(A_i \rightarrow B_i)}$  on the left and something of sort  $\mathcal{K}^{\wedge_{\underline{n}} A_i}$  on the right and as an ion has the sort  $\mathcal{K}^{\wedge_{\underline{n}} B_i}$ . This is meant to represent a derivation  $\Gamma \vdash t u : \wedge_{\underline{n}} B_i$  which, from Lemma 9.38, is built from a set of derivations  $\Gamma \vdash t : A_i \rightarrow B_i$  and  $\Gamma \vdash u : A_i$  for all  $i \in \underline{n}$ . From this set, we can derive  $\Gamma \vdash t : \wedge_{\underline{n}}(A_i \rightarrow B_i)$  and  $\Gamma \vdash u : \wedge_{\underline{n}} A_i$  which matches the sorts of the contents of the application node. Finally, if an ion has the sort  $\mathcal{K}^{A \wedge B}$  then its outer sort can be inflated to  $\mathcal{K}^A$  or  $\mathcal{K}^B$ , matching the  $\wedge \text{E}$  rule.

**Definition 11.55** ( $'\Lambda\text{BIG}^{\text{ks}}$ ). *The Brs  $'\Lambda\text{BIG}^{\text{ks}}$  is the reactive system over the full subcategory of  $\text{BIG}_h(\Sigma_K)$ , the kind sorted Brs over  $\mathcal{K}^\wedge$  with semi-rigid maximum capacities, where the set of interface kinds is generated from  $\text{Int}(\mathcal{K}^\wedge)$  and the reaction rules are as depicted in Figure 11.16*

The site numbering in Figure 11.16 is omitted but is similar to Figure 11.6.

**Definition 11.56** ( $'\Lambda\text{BIG}^{\text{ks}_1}$ ). *The Brs  $'\Lambda\text{BIG}^{\text{ks}_1}$  is defined by taking the largest full subcategory of  $'\Lambda\text{BIG}^{\text{ks}}$  where if  $\text{kind}(r)(\text{def}^A) > 0$  then  $\text{kind}(r)(\text{def}^A) = 1$  and where capacities of places are not zero or  $\odot$ .*


 Figure 11.15: Ion schema for  $\lambda\text{BIG}^\wedge$ 

 Figure 11.16: Parametric reaction rule schema for  $\lambda\text{BIG}^\wedge$

|                                                              |                      |                                                                                                                                                                                                                                                |
|--------------------------------------------------------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $A^{\wedge_{\underline{n}} A_i, \wedge_{\underline{n}} B_i}$ | – Redex and reactum: | $\langle 2, (\mathcal{K}^{\wedge_{\underline{n}} B_i}, \mathcal{K}^{\wedge_{\underline{n}} A_i}), \{x\}, \{x : \wedge_{\underline{n}} A_i \nu\} \rangle \rightarrow \langle 1, \mathcal{K}^{\wedge_{\underline{n}} B_i}, \emptyset, 0 \rangle$ |
| $C^A$                                                        | – Redex:             | $\langle 1, \mathcal{K}^A, \emptyset, 0 \rangle \rightarrow \langle 2, (\mathcal{K}^A, \text{def}^A), \{x\}, \{x : A \delta\} \rangle$                                                                                                         |
|                                                              | – Reactum:           | $\langle 2, (\mathcal{K}^A, \mathcal{K}^A), \emptyset, 0 \rangle \rightarrow \langle 2, (\mathcal{K}^A, \text{def}^A), \{x\}, \{x : A \delta\} \rangle$                                                                                        |
| $D^{A,B}$                                                    | – Redex:             | $\langle 2, (\mathcal{K}^B, \mathcal{K}^A), \emptyset, 0 \rangle \rightarrow \langle 1, \mathcal{K}^B, \emptyset, 0 \rangle$                                                                                                                   |
|                                                              | – Reactum:           | $\langle 1, \mathcal{K}^B, \emptyset, 0 \rangle \rightarrow \langle 1, \mathcal{K}^B, \emptyset, 0 \rangle$                                                                                                                                    |

**Figure 11.17:** Interfaces of the reaction rules

**Definition 11.57** ( $'\Lambda\text{BIG}^s$ ). The Brs  $'\Lambda\text{BIG}^s$  is defined by applying the  $\sigma\lambda\nu\delta$ -sorting to  $'\Lambda\text{BIG}^{\text{ks}_1}$ .

**Definition 11.58** ( $'\Lambda\text{BIG}^{\text{ps}}$ ). The Brs  $'\Lambda\text{BIG}^{\text{ps}}$  is defined by applying a plain sorting to  $'\Lambda\text{BIG}^s$  where the set of name sorts is the set of intersection types.

**Definition 11.59** ( $'\Lambda\text{BIG}^\cap$ ). The Brs  $'\Lambda\text{BIG}^\cap = '\Lambda\text{BIG}^s \times_{'\Lambda\text{BIG}^+} '\Lambda\text{BIG}^{\text{ps}}$  is defined as the paired sorting of  $'\Lambda\text{BIG}^s$  and  $'\Lambda\text{BIG}^{\text{ps}}$ . The ions and reaction rules of  $'\Lambda\text{BIG}^\cap$  are sorted as depicted in Figures 11.15 and 11.16 respectively<sup>10</sup>.

The interfaces of the reaction rules are given in Figure 11.17.

**Lemma 11.60.**  $'\Lambda\text{BIG}^{\text{ks}}$  is a bounded complete and controlled meet full subcategory.

*Proof.* Proposition 11.47 and Lemmas 11.53.1 and 11.53.8 prove that  $'\Lambda\text{BIG}^\cap$  is bounded complete. It is full and controlled by definition. Lemma 11.53.6 proves that it is a meet subcategory.  $\square$

**Corollary 11.61.**  $'\Lambda\text{BIG}^{\text{ks}}$  creates RPOs and weakly reflects pushouts.

*Proof.* RPO creation follows from Lemma 11.60 and Proposition 5.37. Weak pushout reflection follows from Lemma 11.60 and Corollary 5.46.  $\square$

**Proposition 11.62.**  $\mathcal{U}^{\text{ks}_1}$  creates RPOs and reflects pushouts.

*Proof.* Follows similarly to Proposition 11.16.  $\square$

**Proposition 11.63.** The functor  $\mathcal{U}^{\text{ks}} \circ \mathcal{U}^{\text{ks}_1} \circ \mathcal{U}^s \circ \pi_2$  creates RPOs and weakly reflects pushouts.

*Proof.* Follows similarly to Proposition 11.17.  $\square$

<sup>10</sup>Again, we omit the  $\sigma\lambda\nu\delta$ -sorts in the figures for clarity.

**Corollary 11.64.** *Wide bisimilarity over the standard transition system ST for  $\lambda\text{BIG}^\cap$  is a congruence and mono bisimilarity is a congruence for mono contexts.*

As opposed to the partitioned fitting subcategory we used in the last section, this time we use a bounded complete, controlled, meet full subcategory. The use of a full s-category allows us to sort a bigraph with a multitude of interface sorts. This may seem inelegant but in fact it fits nicely with our encoding of intersection type derivations. In the model, an inflation represents multiple uses of the  $\wedge \text{I}$  and  $\wedge \text{E}$  rules. More to the point, it represents the rule

$$\frac{\Gamma \vdash t : A \quad A \ll B}{\Gamma \vdash t : B}$$

found in many intersection type systems [50]. We therefore claim that our encoding is not inelegant as regards the outer sorts; any term can be written as a composition of a term with a least sort (found via set union and guaranteed by Proposition 11.47) and an inflation. This represents a derivation for a term consisting of a derivation of the lesser sort followed by an application of the rule above.

### 11.5.3 Static correspondence

We now encode intersection-typed derivations of  $\Lambda_{\text{sub}}$  terms in  $\lambda\text{BIG}^\cap$ . The encoding is based on Lemma 9.38 which it uses to reason about the types of sub-derivations in order to inductively construct a bigraph.

**Definition 11.65** (encoding type derivations as bigraphs). *The encoding  $\llbracket - \rrbracket_{\Gamma'}$  which takes an intersection typed  $\Lambda_{\text{sub}}$  derivation  $\Gamma \vdash t : A$  with  $\Gamma \subseteq \Gamma'$  to a prime, ground bigraph  $G : \epsilon \rightarrow \{1, \mathcal{K}^A, \text{dom}(\Gamma'), \Gamma'\}$  is defined inductively on the inference of  $\Gamma \vdash t : A$  according to Lemma 9.38 and presented in Figure 11.18.*

The encoding is not injective. For example; given the environment  $x : A \wedge B$ , we can derive  $x : A \wedge B$  using either an application of (**axiom**) or by deriving  $x : A$  and  $x : B$  via (**axiom**) and ( $\wedge \text{E}$ ) and then deriving  $x : A \wedge B$  using ( $\wedge \text{I}$ ). However, both derivations yield the same encoding. This is fine as the second derivation includes some trivial rule applications. Similarly, a translation  $\llbracket - \rrbracket_{\Gamma}$  from (a suitable subcategory of)  $\lambda\text{BIG}^\cap$  to the derivation of terms of  $\Lambda_{\text{sub}}$  typable under the intersection type system could not be surjective. We would hope to define such a translation so that the composition  $\llbracket - \rrbracket_{\Gamma} \circ \llbracket - \rrbracket_{\Gamma}$  would take a derivation to the ‘same’ derivation except without trivial applications and such that the composition  $\llbracket - \rrbracket_{\Gamma} \circ \llbracket - \rrbracket_{\Gamma}$  was the identity up to  $\sigma\lambda\nu\delta$ -name sorts<sup>11</sup>. We defer the translation until we introduce an axiomatisation of kind sortings with semi-rigid capacities and remove the remaining junk bigraphs. This removal will restrict  $\lambda\text{BIG}^\cap$  to

<sup>11</sup>Encodings only have name sorts  $\nu$  – the lesser sort of  $\nu$  and  $\delta$  – in their outer interfaces.



| Derivation                                    | Encoding with index $\Gamma'$ (all variables in $\Gamma'$ have $\sigma\lambda\nu\delta$ -sort $\nu$ )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\Gamma, x : B \vdash x : A$                  | $\uparrow\mathcal{K}^A \circ (\text{var}_{x:B,\nu}^B \oplus \Gamma' / \{x : B, \nu\})$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| $\Gamma \vdash \lambda x.t : A$               | $\uparrow\mathcal{K}^A \circ (\text{lam}_{(x:\wedge_{\underline{n}}A_i, \lambda)}^{\wedge_{\underline{n}}(A_i \rightarrow B_i)} \oplus \text{id}_{\Gamma'}) \llbracket \Gamma, x : \wedge_{\underline{n}}A_i, \nu \vdash t : \wedge_{\underline{n}}B_i \rrbracket_{\Gamma', x:\wedge_{\underline{n}}A_i, \nu}$                                                                                                                                                                                                                                                                                                                                                                                               |
| $\Gamma \vdash \lambda x.t : B \rightarrow C$ | $(\text{lam}_{(x:B, \lambda)}^{B \rightarrow C} \oplus \text{id}_{\Gamma'}) \llbracket \Gamma, x : B, \nu \vdash t : C \rrbracket_{\Gamma', x:B, \nu}$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| $\Gamma \vdash t u : A$                       | $\uparrow\mathcal{K}^A \circ (\text{app}^{\wedge_{\underline{n}}(B_i \rightarrow A_i), \wedge_{\underline{n}}B_i} \oplus (\text{id}_{\Gamma'} \mid \text{id}_{\Gamma'}))$<br>$\circ (\llbracket \Gamma \vdash t : \wedge_{\underline{n}}(B_i \rightarrow A_i) \rrbracket_{\Gamma'} \parallel \llbracket \Gamma \vdash u : \wedge_{\underline{n}}B_i \rrbracket_{\Gamma'})$                                                                                                                                                                                                                                                                                                                                   |
| $\Gamma \vdash t[x/u] : A$                    | $\uparrow\mathcal{K}^A \circ (\text{sub}_{(x:\wedge_{\underline{n}}B_i, \delta)}^{\wedge_{\underline{n}}A_i, \wedge_{\underline{n}}B_i} \oplus (\text{id}_{\Gamma'} \mid \text{id}_{\Gamma'})) \circ \{x : \wedge_{\underline{n}}B_i, \delta / x : \wedge_{\underline{n}}B_i, \nu \ \bar{y} : \wedge_{\underline{n}}B_i, \delta\}$<br>$\circ (\llbracket \Gamma, x : \wedge_{\underline{n}}B_i, \nu \vdash t : \wedge_{\underline{n}}A_i \rrbracket_{\Gamma', x:\wedge_{\underline{n}}B_i, \nu}$<br>$\parallel (\text{def}_{\bar{y}:\wedge_{\underline{n}}B_i, \delta}^{\wedge_{\underline{n}}B_i} \oplus \text{id}_{\Gamma'}) \llbracket \Gamma \vdash u : \wedge_{\underline{n}}B_i \rrbracket_{\Gamma'})$ |

 Figure 11.18: Encoding of intersection typed  $\Lambda_{\text{sub}}$  terms into  $'\Lambda_{\text{BIG}}^\cap$ 

bigraphs which represent parallel compositions of  $\Lambda_{\text{sub}}^\cap$  derivations and the axiomatisation should make the translation and related proofs simpler.

This encoding also preserves types; if  $\Gamma \vdash t : A$  then  $\text{sort} \llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'} = \mathcal{K}^A$ .

**Proposition 11.66.** *The encoding  $\llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'}$  of an intersection typed  $\Lambda_{\text{sub}}$  derivation is well-sorted with outer kind  $\text{sort } \mathcal{K}^A$  and link sort  $\Gamma'$ .*

*Proof.* See Appendix C.4.2, Proposition C.13. □

**Proposition 11.67.** *If  $t \equiv_\alpha u$ ,  $\Gamma \vdash t : A$ , and  $\Gamma \subseteq \Gamma'$  then  $\llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'} \simeq \llbracket \Gamma \vdash u : A \rrbracket_{\Gamma'}$ .*

*Proof.* Induct over the structure of  $t$  where  $\lambda x.t \equiv_\alpha \lambda y.t\{y/x\}$ ,  $y \notin \text{FV}(t)$  and  $t[x/u] \equiv_\alpha t\{y/x\}[y/u]$ ,  $y \notin \text{FV}(t)$  are the non-trivial cases. □

#### 11.5.4 Dynamic properties

**Proposition 11.68** (reaction matches reduction).  *$\llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'} \rightarrow g$  if and only if  $t \rightarrow_{\text{bcgc}} t'$  for some  $t'$  such that  $\llbracket \Gamma \vdash t' : A \rrbracket_{\Gamma'} \simeq g$ .*

Again, we introduce a forgetful functor  $\mathcal{U}^\cap$  between  $\Lambda\text{BIG}^\cap$  and  $\Lambda\text{BIG}^{\text{ut}}$ . It is defined as in Definition 11.37. We compose  $\mathcal{U}^\cap$  with the encoding  $\llbracket \cdot \rrbracket_{\Gamma'}$  of intersection typed  $\Lambda_{\text{sub}}$  derivations to retrieve the encoding  $\llbracket \cdot \rrbracket_{\text{dom}(\Gamma')}$  of  $\Lambda_{\text{sub}}$  terms into  $\Lambda\text{BIG}^{\text{ut}}$ .

**Proposition 11.69.**  $\mathcal{U}^\cap \circ \llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'} = \llbracket X \vdash t : * \rrbracket$  where  $\text{dom}(\Gamma) = X$ .

*Proof.* Induction over the structure of  $t$ . The cases are sketched below.

$$\begin{aligned}
\mathcal{U}^\cap \circ \llbracket \Gamma, x : B \vdash x : A \rrbracket_{\Gamma'} &= \text{var}_{x:\nu} \oplus X / \{x : \nu\} \\
\mathcal{U}^\cap \circ \llbracket \Gamma \vdash \lambda x.t : A \rrbracket_{\Gamma'} &\stackrel{\text{def}}{=} (\text{lam}_{(x:\lambda)} \oplus \text{id}_X) \circ (\mathcal{U}^\cap \circ \llbracket \Gamma, x : \wedge_{\underline{n}} A_i, \nu \vdash t : \wedge_{\underline{n}} B_i \rrbracket_{\Gamma', x:\wedge_{\underline{n}} A_i, \nu}) \\
\mathcal{U}^\cap \circ \llbracket \Gamma \vdash \lambda x.t : B \rightarrow C \rrbracket_{\Gamma'} &\stackrel{\text{def}}{=} (\text{lam}_{(x:\lambda)} \oplus \text{id}_X) \circ (\mathcal{U}^\cap \circ \llbracket \Gamma, x : B, \nu \vdash t : C \rrbracket_{\Gamma', x:B, \nu}) \\
\mathcal{U}^\cap \circ \llbracket \Gamma \vdash t u : A \rrbracket_{\Gamma'} &= (\text{app} \oplus (\text{id}_X \mid \text{id}_X)) \\
&\quad \circ (\mathcal{U}^\cap \circ \llbracket \Gamma \vdash t : \wedge_{\underline{n}} (B_i \rightarrow A_i) \rrbracket_{\Gamma'} \parallel \mathcal{U}^\cap \circ \llbracket \Gamma \vdash u : \wedge_{\underline{n}} B_i \rrbracket_{\Gamma'}) \\
\mathcal{U}^\cap \circ \llbracket \Gamma \vdash t[x/u] : A \rrbracket_{\Gamma'} &= (\text{sub}_{(x:\delta)} \oplus (\text{id}_X \mid \text{id}_X)) \circ \{x : \delta/x : \nu \ \bar{y} : \delta\} \\
&\quad \circ (\mathcal{U}^\cap \circ \llbracket \Gamma, x : \wedge_{\underline{n}} B_i, \nu \vdash t : \wedge_{\underline{n}} A_i \rrbracket_{\Gamma', x:\wedge_{\underline{n}} B_i, \nu} \\
&\quad \parallel (\text{def}_{\bar{y}:\delta} \oplus \text{id}_X)(\mathcal{U}^\cap \circ \llbracket \Gamma \vdash u : \wedge_{\underline{n}} B_i \rrbracket_{\Gamma'}))
\end{aligned}$$

□

**Proposition 11.70.**  $\Lambda\text{BIG}^{\text{ut}}$  reaction on  $\llbracket \cdot \rrbracket_{\text{dom}(\Gamma)}$ -images strongly simulates  $\Lambda\text{BIG}^\cap$  reaction on  $\llbracket \cdot \rrbracket_\Gamma$ -images along  $\mathcal{U}^\cap$ .

*Proof.* The proof follows by the diagram on the right of Figure 11.13 where  $\Lambda_{\text{sub}}^\cap$  denotes the subcalculus of  $\Lambda_{\text{sub}}$ , closed under reduction, of terms typable under the intersection type discipline. Proposition 11.69 proves the squares, Proposition 11.68 proves the top face of the cube, and the matching of  $\Lambda_{\text{sub}}$  reduction by  $\Lambda\text{BIG}^{\text{ut}}$  reaction (Proposition 11.20) proves the bottom face. □

**Proposition 11.71.**  $\Lambda\text{BIG}^\cap$  can simulate  $\beta$ -reduction, is confluent on terms without metavariables, preserves strong normalisation of  $\beta$ -reduction, and has full composition of substitutions on encodings of intersection typed  $\Lambda_{\text{sub}}$  derivations.

We have the following corollary of Theorem 9.51.

**Corollary 11.72.**  $\Lambda\text{BIG}^\cap$  is strongly normalising on encodings of intersection typed  $\Lambda_{\text{sub}}$  derivations.

## 11.6 Conclusions

In this chapter we improved on Milner's model of the  $\lambda$ -calculus by discarding most of the bigraphs which do not correspond to  $\Lambda_{\text{sub}}$  terms. We believe that it is possible to remove all such terms by

first sorting out the bigraphs where **def**-nodes are chained in cycles and then iteratively wrapping any bigraphs with outer **def**-nodes inside **sub**-nodes using some order *e.g.* leftmost **def**-nodes first. This translation should hopefully be deterministic up to a reordering of independent substitutions.

Our main contribution in this chapter is a demonstration of how kind sorting can be used to model type derivation trees for some basic type disciplines, allowing us to model simply-typed and intersection-typed  $\lambda$ -calculi. The latter represents a bigraphical encoding of all strongly normalising terms and parallel compositions thereof and, for a large subcategory of the Brs, only these terminating terms are represented. We conjecture that all bigraphs in both  $\mathsf{ABIG}^{\rightarrow}$  and  $\mathsf{ABIG}^{\cap}$  terminate although this is based purely on intuition. Rather than pursue this directly, we would prefer to address the deficiency of the sorting which would make such a proof much simpler.

The  $\sigma\lambda\nu\delta$ -sorting combines a simple link sorting with conditions involving both place and link graphs. It is reasonably complex considering its intention and its use seems rather limited to our application. However, it demonstrates that complex sortings may be safely applied to bigraphs.

In one of the earliest definitions of bigraphs [104], Milner posed the following question regarding typing of bigraphs:

*“Analogous to signing ports, we can type them.... The practical significance is obvious. Again we get a safe forgetful functor from typed to untyped BRSs. However, the situation is only simple of [sic] we demand exact matching of types: no polymorphism and no subtyping. It is an intriguing question how to deal with these richer type phenomena in bigraphical systems.” [104]*

It appears that our model of intersection typed  $\lambda$ -calculus (representing finite polymorphism) and Bundgaard and Sassone’s link subsorting for the polyadic  $\pi$ -calculus [24] have provided some answers to this question.

We believe that the models of typed  $\lambda$ -calculi present a significant theoretical application for kind sortings and a strong case for the investigation of safe subcategories of sortings.

### 11.6.1 Further work

We have modelled the simply typed and intersection typed  $\lambda$ -calculi with bigraphs using relatively simple sortings<sup>12</sup>. This raises the questions of whether a simpler sorting would suffice and what the limitation of kind sorting is with respect to modelling type systems. The latter question could be investigated by attempting to model other simple types (products, coproducts) or recursion

<sup>12</sup>We should note that Milner’s idea of multi-nodes [111] was crucial to our models.

although we believe that our approach is limited. It may be beneficial to examine encodings of typed  $\lambda$ -calculi in the  $\pi$ -calculus [144, 136]. Appendix C.5 presents a related idea of deriving typed  $\lambda$ BIG terms using a set of typing rules.

Grohmann and Miculan’s directed bigraphs marry both Milner’s (*output-linear*) link graphs and Sassone and Sobociński’s (*input-linear*) linking structures [138] whilst retaining RPOs [63]. Their generalisation admits many new linking patterns for bigraphs and allows an encoding, using the new input-linearity, of the Fusion calculus of Parrow and Victor [126] in bigraphs [64]. A sound and complete axiomatisation of these bigraphs is also provided.

Models of both the call-by-name and call-by-value  $\lambda$ -calculi have also been presented with directed bigraphs [63]. However, this generalisation of link graphs does not seem able to express locality of names and hence name-scoping. In that respect, Milner’s modelling of the  $\lambda$ -calculus seems preferable and, we believe, removes more junk bigraphs. Furthermore, the modelling of call-by-name and call-by-value reductions does not seem dependent on the linking structure but rather on the place graph and the activity of controls *i.e.* the directed links are incidental to the models. That said, we believe the name-scoping limitation to be minor; it seems straightforward to add locality of names to directed bigraphs. Directed bigraphs combine easily with kind sortings, allowing us to sort these  $\lambda$ -models.

While Grohmann and Miculan concentrate on encodings of pure  $\lambda$ -terms, the model is still based on  $\Lambda_{\text{sub}}$  where reduction matches reaction. Therefore, the results of Part II can be used to reason about normalisation properties of encodings of  $\Lambda_{\text{sub}}$ -terms in their models.

Possible future work in this direction includes applying our sorting scheme and interpretation of typing to models of call-by-name, call-by-value, or lazy  $\lambda$ -calculi. Sangiorgi [135] investigated connections between behavioural equivalences of Milner’s model [100] of Abramsky’s lazy  $\lambda$ -calculus [4] and operational semantics provided by Levi-Longo Trees [95, 124] and applicative bisimulation [3]. It is worth investigating the equivalence induced by the bisimilarity (in the Brss) of encodings of (typed)  $\lambda$ -terms to see how it compares to the above equivalences and if, or how, our removal of malformed bigraphs from  $\lambda$ BIG affects bisimulation. More related work in this area includes Gordon’s study of I/O using a typed  $\lambda$ -calculus [62] and Tiuryn and Wand’s observational equivalence for an extended pure  $\lambda$ -calculus [143].

Simply typed versions of  $\lambda\text{lr}$ ,  $\lambda\text{es}$ , and  $\Lambda_{\text{sub}}$  have all been put in relation to the multiplicative exponential fragment of linear logic (MELL) via proof nets [79, 78, 81].  $\lambda\text{BIG}^\rightarrow$  models simple type derivations for  $\Lambda_{\text{sub}}$  with a partial static correspondence. This suggests a translation from place-linear  $\lambda\text{BIG}^\rightarrow$  bigraphs to proof nets via  $\Lambda_{\text{sub}}$  type derivations and would create a further link

between bigraphs and interaction nets. However, our intuition is that the cited relationship between  $\Lambda_{\text{sub}}$  and MELL proof nets (which composes the translation  $\mathcal{T}$  from  $\Lambda_{\text{sub}}$  to  $\lambda\text{es}$  of Section 9.3 with the translation from  $\lambda\text{es}$  to proof nets) may not be revealing as the copying of substitutions by  $\mathcal{T}$  introduces unwanted detail. A better translation should be investigated.

### 11.6.2 Related work

Appendix C.3 discusses some of the similarities between  $\Lambda\text{BIG}$  and non-bigraphical calculi. Appendix C.5.1 contains some thoughts about the models of  $\lambda$ -calculi and the relationship between the type and interface sort preorders.

Our combination of kind sorting and  $\sigma\lambda\nu\delta$ -sorting bears some similarities with Bundgaard and Hildebrandt's sorting for the bigraphical model of Higher-Order Mobile Embedded Resources (Homer), a higher-order process calculus, with explicit substitutions (Homer $\sigma$ ) [22, 23]. They model a typed version of Homer with explicit substitutions, Homer $\sigma$ . However, the type of a Homer term in this case is a set of names which contains the free names of the term. The typing is used to ensure that reduction in Homer $\sigma$  does not lose free names so that reduction can match reaction in the bigraphical model (where the notion of (outer) interface preservation is implicit in the dynamics). This differs from the typing we are trying to model in this chapter. Bundgaard and Hildebrandt posed the question of whether the sorting used for Homer $\sigma$  could be expressed in any other sorting scheme. Kind sorting is able to express the placing restrictions of their sorting, using the kind sorting with rigid capacities represented in Figure 11.19. In the figure, the labels on edges specify that nodes of the target control will contain exactly one node of the source control in each bigraph. Our  $\sigma\lambda\nu\delta$ -sorting also shares some similarities to their sorting. However, it is not clear to us how they sort out bigraphs where children of **def**-nodes have the same link as the **def**-node. Our combined sorting differs from theirs as it is preserved by composition; this is true for Homer $\sigma$  but only vacuously so as their sorting only admits ground bigraphs and so composition of sorted bigraphs is impossible.

Bundgaard and Sassone have modelled a typed polyadic pi-calculus in bigraphs [24] where the type system was based on Pierce and Sangiorgi's capability types [128] with subsorting. They introduce a new link sorting, *subsorting*, which represents the first step in a theory of link-subtyping with binding for bigraphs. The approach differs from previous work as the edges of bigraphs are sorted rather than the ports. The sort of a node is then derived from the sorts of the links to which it is connected. As a result, their presentation only uses one control to model an input/output prefix of some message length. In comparison, this paper has introduced a place-sorting where the

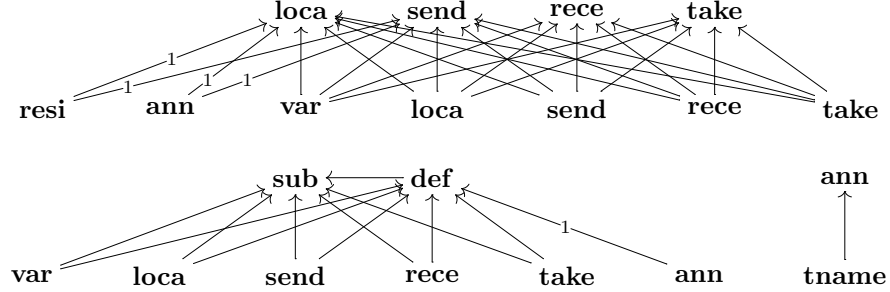


**Figure 11.19:** Representation of the place graph sorting of  $\text{Homersigma}$  as a kind sorting with rigid capacities

controls are sorted and where we have one control per type per lambda constructor.

We have demonstrated that kind sorting can be used for modelling the  $\lambda$ -calculus with intersection types by restricting the parent-child relationships of the place graph. We do not claim that this provides the best model. For example, other alternatives may involve a more sophisticated link sorting. In this vein, Bundgaard and Sassone's work on modelling the typed polyadic  $\pi$ -calculus [24] in bigraphs seems the most promising to date. Their *subsorting* on the link structure is based on a set  $\mathcal{S}$  with a preorder  $\leq$  with *bounded meets*, the dual notion of bounded complete. Their sorting creates RPOs and weakly reflects pushouts.

Both the preorder  $(\tau, \ll)$  and the partial order  $(\text{Int}(\mathcal{K}^\wedge), \subseteq)$  (see pages 40 and 232 respectively) have the property of bounded meets (up to  $\sim$  for the preorder). In fact, they have all meets; for any two interface sorts  $\mathcal{K}^A$  and  $\mathcal{K}^B$ , their meet is  $\mathcal{K}^A \cap \mathcal{K}^B = \mathcal{K}^{A \wedge B}$ . The bounded meets property was sufficient in Bundgaard and Sassone's proof of weak pushout reflection and it is the same for our model. On the other hand, we have shown that a bounded complete property is sufficient for the proof of RPO creation in  $\lambda\text{BIG}^\cap$ . It would seem that RPO creation and strong (resp. weak) pushout reflection hold for kind fitting (resp. full) s-categories depending on the structure of unions of interface sorts and intersections of control and interface sorts. Bounded completeness is important for RPO creation; a meet structure is important in full s-categories to ensure weak pushout reflection. Therefore, it seems advisable to study subcategories of kind sortings where the partial order over interface and control sorts with respect to subset inclusion forms a lattice or semi-lattice structure or at least satisfies bounded completeness or its dual. A special case is the partitioned s-categories whose partial order of this type is trivially a lattice.

The property of bounded meets was introduced in the context of type systems for the  $\pi$ -calculus by Hennessy and Riely [68] who extended a previous type system due to Pierce and Sangiorgi [128].

### Summary of Part III

Our objective in the final part of the dissertation was to demonstrate the expressiveness of kind sorting by applying it to model bigraphical algorithms, semi-structured data, and typed calculi.

Our model of typed  $\lambda$ -calculi models type derivations of typable terms rather than the terms themselves. We observe that type derivations may themselves be considered a form of semi-structured data. We succeed in removing most, but not all, malformed bigraphs from the models. The result of this is that the model of the intersection typed  $\lambda$ -calculus represents all, and nearly only, parallel compositions of strongly normalising terms.

# Conclusions



# Chapter 12

## Conclusions

*And I am a writer, writer of fictions  
I am the heart that you call home  
And I've written pages upon pages  
Trying to rid you from my bones*

*The Engine Driver – The Decemberists*

### 12.1 Conclusions

We have presented and investigated sortings on the spatial structure of bigraphs. We expounded on Jensen and Milner's suggestion to define a sorting which enforces a containment relation specified on the set of controls of a signature. We added to this suggestion the notion of invisible controls, which add sub-structure to place graphs which is hidden from outer interfaces. This was used to model Milner's multi-nodes which were then used to add more order to the place graph, allowing the modelling of semi-structured data such as XML. Finally, we added capacities to the spatial structure, allowing us to satisfy the hierarchical constraints of the  $\lambda$ -calculus grammar in our models of  $\lambda$ -calculi.

Our sortings allow an expression of absence in parameters to parametric reaction rules. We gave examples of this, demonstrated how it admits a basic level of flow control to the dynamics of a reactive system, and applied this to the modelling of simple algorithms.

We defined subcategories of the sortings and proved safety properties of both the sortings and particular subcategories. Properties were identified which suffice for RPO creation and varying degrees of pushout reflection and which are based on the lattice structure of the partial order on sorts with respect to subset inclusion, namely bounded completeness and a meet semi-lattice structure. The language of opfibrations, introduced to bigraphs in earnest by Birkedal, Debois, and Hildebrandt, was exploited to present a notion of pushout reflection and labelled transition based on jointly opcartesian bounds. We also defined some simple link sortings and a more complicated sorting specifically aimed at sorting Milner’s model of the  $\lambda$ -calculus.

Bigraphical models of basic typed  $\lambda$ -calculi were presented using a novel sorting scheme which removes most of the bigraphs which do not correspond to type derivations. The interpretation of types as sets of controls matches our intuitions of the respective type preorders; the untyped model has a single element representing the single type, the simply typed model has an infinite number of sets, each of which is unrelated (with respect to subset inclusion), and the intersection type model has an infinite number of sets which are related via intersections of sets.

Given that it was not meant as a particularly interesting explicit substitution calculus in its own right but rather a demonstration of local bigraphs and a case study of confluence in bigraph theory<sup>1</sup>, it is remarkable that Milner’s  $\Lambda_{\text{sub}}$  satisfies so many desirable properties of explicit substitution calculi which traditional calculi do not. We attribute this both to the correctness of the model and to the notion of non-local substitution. Besides our investigation of the properties of  $\Lambda_{\text{sub}}$ , our contribution to this area was the method of simulating non-local substitution with local substitution which allows normalisation properties to be reflected along the simulation. Joint work with Delia Kesner also provided a characterisation of the set of strongly normalising terms of  $\Lambda_{\text{sub}}$  and  $\lambda\text{es}$  via intersection types and a second method of simulation.

Finally, our results for  $\Lambda_{\text{sub}}$  are used with the dynamic correspondence between typed versions of the  $\lambda$ -calculus and their bigraphical encodings to reason about confluence and termination in the bigraphical models and come close to describing a bigraphical model of all and only the parallel compositions of terminating processes.

## 12.2 Future work

At the time of writing, bigraphs are still a relatively new concept. Their core theory has been developed significantly since conception but many questions remain. These are better covered by

<sup>1</sup>This has been communicated to the author by Robin Milner.

the instigators of this research [90, 74, 114] however we have touched upon some of the partially investigated areas in this dissertation.

### Applications of kind sortings

More applications (and limitations) of kind sortings should be investigated.

### Bilogics

The work on spatial logic for process calculi by Cardelli, Gordon, and Caires [33, 34, 26, 27] has been investigated in the bigraphical framework by Conforti, Macedonio, and Sassone [39, 38]. Their logic is composed of a spatial logic and a logic describing connectivity, reflecting the separation of pure bigraphs into place and link graphs. Their presentation takes advantage of Milner’s axiomatisation of pure bigraphs [108].

The axiomatisations of kind bigraphs follow similarly to the pure axiomatisations. Kind sortings add the *verticals*, the identities on placing and linking which ‘inflate’ sorts of places, as basic constructors. The definition of ions is straightforward for most of the sortings besides the kind sorting with min-max capacities where the disparity between the expressiveness of the signature, which allows specification of minimum and maximum capacities, and the interface sorts which only allow exact capacities, complicates matters. Given that kind sorted parametric reaction rules admit some notion of spatial preconditions and that XML data may be better encoded with kind sortings, we believe that a generalisation of bilogics to kind sorted bigraphs would be worth investigating.

### Models of $\lambda$ -calculi

The sorting we present for the  $\lambda$ -calculus still contains a deficiency. We have highlighted this and believe that it can be addressed with a further sorting. Once we have correctly captured the  $\lambda$ -calculus, it could be interesting to investigate behavioural equivalences on the canonical labelled transition system.

Elsewhere, we have jointly proven open confluence for an extension of  $\Lambda_{\text{sub}}$  [81]. Open confluence does not have an immediate translation in bigraphs where reaction is defined on ground terms. However, we believe that we may be able to model metavariables with controls. Consider a non-atomic control **meta** with unbounded capacity which can contain **sub** nodes and atomic metavariables **mvar**. The reaction

$$\text{meta} \circ (\text{mvar}_x \otimes \square_1) \parallel \text{def}_{(x)}(\llbracket t \rrbracket) \rightarrow \text{meta} \circ (\text{sub}_{(y)}(\text{mvar}_y \parallel \text{def}_{(y)}(\llbracket t \rrbracket)) \otimes \square_1) \parallel \text{def}_{(x)}(\llbracket t \rrbracket)$$

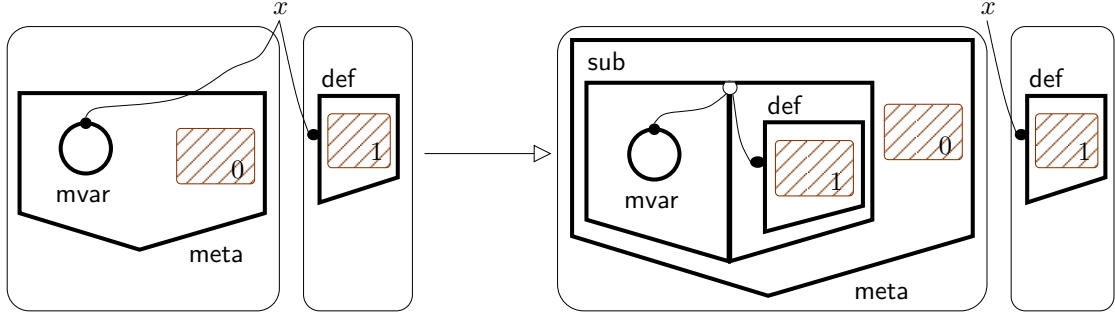


**Figure 12.1:** *Proposed reaction rule to model non-local substitution of metavariables*

depicted in Figure 12.1 exhibits the non-local substitution of metavariables

$$C[\mathbb{X}_\Delta][x/t] \rightarrow C[\mathbb{X}_\Delta[x/t]][x/t], x \in \Delta, x \notin \text{FV}(C)$$

up to  $\simeq$ . The substitution is modelled by enclosing a metavariable inside its own personal closure. This differs slightly from the substitution reduction added to  $\Lambda_{\text{sub}}$  for metavariables [81] where  $\mathbb{X}_x[x/t]$  has no redex at the root however this seems a minor variation.

## Typing

Is our modelling of typed  $\lambda$ -calculus a sensible approach? Perhaps not. Using our approach, we find that *type derivations* are what we naturally model. This in itself is interesting; type derivations could be thought of as a particular semi-structured data. However, we leave the question of whether typed  $\lambda$ -terms can be better represented – our representation presents the internal observer with the whole derivation tree and the external observer with the derived type and the environment (via the outer interface). The internal details of the derivation bloat the signature set and may be unnecessary for some applications. However, we have no good intuitions here.

Bundgaard and Sassone adopt a different approach in their model of typed polyadic calculus [24] but as their sorting is a link sorting, we cannot directly compare both approaches. However, we use a meet semi-lattice to secure (at least) weak pushout reflection and they use a preorder with bounded meets. We find that bounded completeness suffices to secure RPO creation rather than the stronger notion of a join semi-lattice. We conjecture that the property of bounded meets suffices for pushout reflection for jointly opcartesian bounds in subcategories of kind sortings.

The limitations of modelling type systems with kind sortings should be investigated. The type systems we studied have a simple set-based interpretation. We believe that our reliance on sets will be the main limiting factor for our means of representing types.

*And in the end  
The love you take  
Is equal to the love you make*

*The End* – The Beatles

# Bibliography

- [1] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.
- [2] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.
- [3] Samson Abramsky. *Domain theory and the logic of observable properties*. PhD thesis, Queen Mary College, University of London, 1987.
- [4] Samson Abramsky. The lazy lambda calculus. In D. A. Turner, editor, *Research topics in functional programming*, pages 65–116. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
- [5] Zena M. Ariola and Matthias Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3):265–301, 1997.
- [6] Zena M. Ariola and Jan Willem Klop. Lambda calculus with explicit recursion. *Information and Computation*, 139(2):154–233, 1997.
- [7] Steve Awodey. *Category Theory*. Number 49 in Oxford Logic Guides. Oxford University Press, revised edition, 2006.
- [8] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition, 1984.
- [9] Michael Barr and Charles Wells. *Category Theory for Computing Science*. Les Publications CRM, Montréal, third edition, 1999.

- [10] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992.
- [11] Clara Bertolissi. The graph rewriting calculus: confluence and expressiveness. In Mario Coppo, Elena Lodi, and G. Michele Pinna, editors, *9th Italian conference on Theoretical Computer Science (ICTCS 2005), Siena, Italy*, volume 3701 of *Lecture Notes in Computer Science*, pages 113–127. Springer Verlag, October 2005.
- [12] Clara Bertolissi, Paolo Baldan, Horatiu Cirstea, and Claude Kirchner. A rewriting calculus for cyclic higher-order term graphs. *Electronic Notes in Theoretical Computer Science*, 127(5):21–41, 2005.
- [13] Lars Birkedal, Mikkel Bundgaard, Troels C. Damgaard, Søren Debois, Ebbe Elsborg, Arne John Glenstrup, Thomas Troels Hildebrandt, Robin Milner, and Henning Niss. Bi-graphical programming languages for pervasive computing. In Thomas Strang, Vinny Cahill, and Aaron Quigley, editors, *Proceedings of Pervasive 2006 International Workshop on Combining Theory and Systems Building in Pervasive Computing*, pages 653–658, May 2006.
- [14] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bi-graphical Models of Context-aware Systems. Technical Report TR-2005-74, IT-Universitetet, København, Rued Langgards Vej 7, DK-2300 Copenhagen V, November 2005. ISBN: 87-7949-110-3.
- [15] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bi-graphical Models of Context-aware Systems. In *Foundations of Software Science and Computation Structures (FOSSACS) 2006*, number 3921 in *Lecture Notes in Computer Science*, March 2006.
- [16] Lars Birkedal, Søren Debois, and Thomas Hildebrandt. Sortings for reactive systems. In Christel Baier and Holger Hermanns, editors, *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2006.
- [17] Lars Birkedal, Søren Debois, and Thomas Hildebrandt. Sortings for reactive systems. Technical Report TR-2006-84, IT-Universitetet, København, Rued Langgards Vej 7, DK-2300 Copenhagen V, March 2006.
- [18] Roel Bloo. *Preservation of termination for explicit substitution*. PhD thesis, Eindhoven University of Technology, 1997.

- [19] Roel Bloo and Herman Geuvers. Explicit substitution: on the edge of strong normalization. *Theoretical Computer Science*, 211(1–2):375–395, 1999.
- [20] Roel Bloo and Kristoffer Høgsbro Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN-95: Computer Science in the Netherlands*, November 1995.
- [21] Pietro Braione and Gian Pietro Picco. On calculi for context-aware coordination. In Rocco De Nicola, Gian Luigi Ferrari, and Greg Meredith, editors, *COORDINATION*, volume 2949 of *Lecture Notes in Computer Science*, pages 38–54. Springer, 2004.
- [22] Mikkel Bundgaard and Thomas Hildebrandt. Bigraphical semantics of higher-order mobile embedded resources with local names. In Reiko Heckel, Barbara König, and Arend Rensink, editors, *Proceedings of GT-VC '05 (Graph Transformation for Verification and Concurrency)*, number 05–34 in CTIT Technical Reports. Centre for Telematics and Information Technology, University of Twente, 2005.
- [23] Mikkel Bundgaard and Thomas Hildebrandt. Bigraphical semantics of higher-order mobile embedded resources with local names. Technical Report TR-2005-70, IT-Universitetet, København, Rued Langgards Vej 7, DK-2300 Copenhagen V, September 2005.
- [24] Mikkel Bundgaard and Vladimiro Sassone. Typed polyadic pi-calculus in bigraphs. In *Proceedings of the 8th ACM SIGPLAN international conference on Principles and Practice of Declarative Programming 2006*, pages 1–12, 2006.
- [25] Luís Caires. *A Model for Declarative Programming and Specification with Concurrency and Mobility*. PhD thesis, University of Lisbon, 1999.
- [26] Luís Caires and Luca Cardelli. A Spatial Logic for Concurrency (part I). *Journal of Information and Computation*, 186(2):194–235, 2003.
- [27] Luís Caires and Luca Cardelli. A Spatial Logic for Concurrency (part II). *Theoretical Computer Science*, 322(3):517–565, 2004.
- [28] Luca Cardelli. Describing semistructured data. *ACM SIGMOD Record*, 30(4):80–85, 2001.
- [29] Luca Cardelli, Philippa Gardner, and Giorgio Ghelli. A spatial logic for querying graphs. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming*,



- 29th International Colloquium (ICALP 2002), Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 597–610. Springer, 2002.
- [30] Luca Cardelli, Philippa Gardner, and Giorgio Ghelli. Manipulating trees with hidden labels. *Electronic Notes in Theoretical Computer Science*, 172:177–201, 2007.
- [31] Luca Cardelli and Giorgio Ghelli. Tql: a query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 14(3):285–327, 2004.
- [32] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *FoSSaCS '98: Proceedings of the First International Conference on Foundations of Software Science and Computation Structure*, pages 140–155, London, UK, 1998. Springer-Verlag.
- [33] Luca Cardelli and Andrew D. Gordon. Anytime, anywhere: modal logics for mobile ambients. In *POPL '00: Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 365–377, New York, NY, USA, 2000. ACM Press.
- [34] Luca Cardelli and Andrew D. Gordon. Logical properties of name restriction. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications, 5th International Conference, (TLCA 2001), Krakow, Poland, May 2-5, 2001, Proceedings*, volume 2044 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2001.
- [35] Giuseppe Castagna, Jan Vitek, and Francesco Zappa Nardelli. The seal calculus. *Information and Computation*, 201(1):1–54, 2005.
- [36] Alonzo Church. *The Calculi of Lambda Conversion*. Princeton University Press, Princeton, NJ, 1941. Reprinted 1963 by University Microfilms, Ann Arbor, MI.
- [37] Alonzo Church and J. Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936.
- [38] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. Bigraphical logics for XML. In Andrea Calì, Diego Calvanese, Enrico Franconi, Maurizio Lenzerini, and Letizia Tanca, editors, *SEBD*, pages 392–399, 2005.
- [39] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. Spatial logics for bigraphs. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Proceedings of the 32th International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 766–778. Springer, 2005.

- [40] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for  $\lambda$ -terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19:139–156, 1978.
- [41] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the  $\lambda$ -calculus. *Notre-Dame Journal of Formal Logic*, 21(4):685–693, October 1980.
- [42] P.-L. Curien. Categorical combinators. *Information and Control*, 69(1–3):188–254, 1986.
- [43] Haskell Curry and Robert Feys. *Combinatory Logic*. Studies in Logic and the Foundations of Mathematics. North-Holland, 1958. Volume II, with Jonathan Seldin, 1972.
- [44] Silvano Dal Zilio, Denis Lugiez, and Charles Meyssonnier. A logic you can count on. In *POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 135–146, New York, NY, USA, 2004. ACM Press.
- [45] Troels C. Damgaard and Lars Birkedal. Axiomatizing binding bigraphs. *Nordic Journal of Computing*, 13(1):58–77, 2006.
- [46] René David and Bruno Guillaume. A  $\lambda$ -calculus with explicit weakening and explicit substitution. *Mathematical Structures in Computer Science*, 11(1):169–206, 2001.
- [47] Søren Debois and Troels C. Damgaard. Bigraphs by Example. Technical Report TR-2005-61, IT-Universitetet, København, Rued Langgards Vej 7, DK-2300 Copenhagen V, October 2005. ISSN: 1600-6100.
- [48] Nachum Dershowitz. A note on simplification orderings. *Information Processing Letters*, 9(5):212–215, 1979.
- [49] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 243–320. Elsevier and MIT Press, 1990.
- [50] Mariangiola Dezani-Ciancaglini, Furio Honsell, and Fabio Alessi. A complete characterization of complete intersection-type preorders. *ACM Transactions on Computational Logic*, 4(1):120–147, 2003.
- [51] Roberto Di Cosmo and Delia Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets (extended abstract). In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, pages 35–46, Warsaw, July 1997.

- [52] Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof nets and explicit substitutions. *Mathematical Structures in Computer Science*, 13(3):409–450, 2003.
- [53] Dan Dougherty and Pierre Lescanne. Reductions, intersection types, and explicit substitutions. *Mathematical Structures in Computer Science*, 13(1):55–85, 2003.
- [54] Hartmut Ehrig and Julia Padberg. Graph grammars and Petri net transformations. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 496–536. Springer, 2003.
- [55] Maribel Fernández and Ian Mackie. Closed reductions in the  $\lambda$ -calculus. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic*, volume 1683 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 1999.
- [56] Maribel Fernández, Ian Mackie, and François-Régis Sinot. Interaction nets vs. the  $\rho$ -calculus: Introducing bigraphical nets. *Electronic Notes in Theoretical Computer Science*, 154(3):19–32, 2006.
- [57] Cédric Fournet and Georges Gonthier. The reflexive cham and the join-calculus. In *POPL '96: Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 372–385, New York, NY, USA, 1996. ACM.
- [58] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer, 1996.
- [59] Martin Gardner. Mathematical games: The fantastic combinations of John Conway’s new solitaire game ‘Life’. *Scientific American*, 223(4):120–123, October 1970. The original description of Conway’s game of LIFE.
- [60] Philippa Gardner and Lucian Wischik. Explicit fusions. In *MFCS '00: Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 373–382, London, UK, 2000. Springer-Verlag.
- [61] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
- [62] Andrew D. Gordon. *Functional programming and input/output*. Cambridge University Press, New York, NY, USA, 1994.

- [63] Davide Grohmann and Marino Miculan. Directed bigraphs. *Electronic Notes in Theoretical Computer Science*, 173:121–137, April 2007.
- [64] Davide Grohmann and Marino Miculan. Reactive systems over directed bigraphs. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 380–394. Springer, 2007.
- [65] Davide Grohmann and Marino Miculan. An algebra for directed bigraphs. *Electronic Notes in Theoretical Computer Science*, 203(1):49–63, 2008.
- [66] John Paulin Hansen et al. Delca project.
- [67] Thérèse Hardin. Confluence results for the pure strong categorical logic C.C.L.  $\lambda$ -calculi as subsystems of C.C.L. *Theoretical Computer Science*, 65(3):291–342, 1989.
- [68] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.
- [69] Hugo Herbelin. *Séquents qu'on calcule : de l'interprétation du calcul des séquents comme calcul de  $\lambda$ -termes et comme calcul de stratégies gagnantes*. Thèse d'université, Université Paris 7, Janvier 1995.
- [70] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
- [71] Gerard Huet and Derek C. Oppen. Equations and rewrite rules: A Survey. Technical Report CS-TR-80-785, Stanford University, Department of Computer Science, January 1980.
- [72] Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.
- [73] Ole Høgh Jensen. *Mobile Processes in Bigraphs*. PhD thesis, Department of Computer Science, Aalborg University, October 2006.
- [74] Ole Høgh Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, Computer Laboratory, University of Cambridge, February 2004.
- [75] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986.

- [76] Fairouz Kamareddine and Alejandro Ríos. Extending a  $\lambda$ -calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4), July 1997.
- [77] Delia Kesner. Confluence properties of extensional and non-extensional  $\lambda$ -calculi with explicit substitutions (extended abstract). In Harald Ganzinger, editor, *RTA*, volume 1103 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1996.
- [78] Delia Kesner. The theory of calculi with explicit substitutions revisited. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic: 21st International Workshop (CSL 2007), 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11–15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 238–252. Springer, 2007.
- [79] Delia Kesner and Stéphane Lengrand. Resource operators for  $\lambda$ -calculus, 2004. Available at <http://www.pps.jussieu.fr/~kesner/papers/>.
- [80] Delia Kesner and Stéphane Lengrand. Extending the explicit substitution paradigm. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference (RTA 2005), Nara, Japan, April 19–21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 407–422. Springer, 2005.
- [81] Delia Kesner and Shane Ó Conchúir. Fundamental properties of Milner’s non-local explicit substitution calculus, 2007.
- [82] Jan Willem Klop. *Combinatory Reduction Systems. PhD Thesis*, volume 127 of *Mathematical Centre Tracts*. CWI, Amsterdam, 1980.
- [83] Jan Willem Klop. Term Rewriting Systems. In Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, Inc., New York, NY, USA, 1992.
- [84] Jan Willem Klop. Term graph rewriting. In Gilles Dowek, Jan Heering, Karl Meinke, and Bernhard Möller, editors, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 1074 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1995.
- [85] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In John Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Peramon Press, 1970.

- [86] Yves Lafont. Interaction nets. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 95–108, New York, NY, USA, 1990. ACM Press.
- [87] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 1986.
- [88] F. William Lawvere and Robert Rosebrugh. *Sets for Mathematics*. Cambridge University Press, 2003.
- [89] F. William Lawvere and Stephen H. Schanuel. *Conceptual Mathematics*. Cambridge University Press, 1998. A first introduction to categories.
- [90] James J. Leifer. *Operational congruences for reactive systems*. PhD thesis, Computer Laboratory, University of Cambridge, 2001. Available in revised form as Technical Report 521, Computer Laboratory, University of Cambridge, 2001, from <http://pauillac.inria.fr/~leifer/research.html>.
- [91] James J. Leifer and Robin Milner. Deriving bisimulation congruences for reactive systems. In Catuscia Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2000.
- [92] James J. Leifer and Robin Milner. Transition systems, link graphs and petri nets. Technical Report UCAM-CL-TR-598, Computer Laboratory, University of Cambridge, August 2004.
- [93] Stéphane Lengrand. Induction principles as the foundation of the theory of normalisation: Concepts and techniques. Technical report, PPS laboratory, Université Paris 7, March 2005. available at <http://hal.ccsd.cnrs.fr/ccsd-00004358>.
- [94] Stéphane Lengrand, Pierre Lescanne, Dan Dougherty, Mariangiola Dezani-Ciancaglini, and Steffen van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.
- [95] Giuseppe Longo. Set-theoretical models of  $\lambda$ -calculus: theories, expansions, isomorphisms. *Annals of Pure and Applied Logic*, 24:153–188, August 1983.
- [96] Mícheál Mac an Airchinnigh. The confederation of the isles. Example scenario for course material and exam questions, 2002.
- [97] Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, second edition, 1998.

- [98] Paul-André Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications, Edinburgh*, number 902 in Lecture Notes in Computer Science, pages 328–334, 1995.
- [99] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.
- [100] Robin Milner. Functions as processes. Technical Report RR-1154, INRIA Sophia-Antipolis, February 1990.
- [101] Robin Milner. The polyadic  $\pi$ -calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.
- [102] Robin Milner. Calculi for interaction. *Acta Informatica*, 33(8):707–737, 1996.
- [103] Robin Milner. *Communicating and mobile systems: the  $\pi$ -calculus*. Cambridge University Press, New York, NY, USA, 1999.
- [104] Robin Milner. Bigraphical reactive systems: basic theory. Technical Report UCAM-CL-TR-523, Computer Laboratory, University of Cambridge, 2001.
- [105] Robin Milner. Computational flux. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 220–221, New York, NY, USA, 2001. ACM Press.
- [106] Robin Milner. Bigraphs whose names have multiple locality. Technical Report UCAM-CL-TR-603, Computer Laboratory, University of Cambridge, September 2004.
- [107] Robin Milner. Local bigraphs, confluence and  $\lambda$ -calculus (early draft). Technical Report to appear, Computer Laboratory, University of Cambridge, 2004.
- [108] Robin Milner. Axioms for bigraphical structure. *Mathematical Structures in Computer Science*, 15(6):1005–1032, 2005.
- [109] Robin Milner. Bigraphs: A tutorial, 2005.
- [110] Robin Milner. Pure bigraphs. Technical Report UCAM-CL-TR-614, Computer Laboratory, University of Cambridge, 2005.

- [111] Robin Milner. Local bigraphs and confluence: two conjectures. In Roberto Amadio and Iain Phillips, editors, *Proceedings of the 13th International Workshop on Expressiveness in Concurrency (EXPRESS '06)*. Electronic Notes in Theoretical Computer Science, 2006.
- [112] Robin Milner. Local bigraphs, confluence and  $\lambda$ -calculus. Technical Report to appear, Computer Laboratory, University of Cambridge, 2006.
- [113] Robin Milner. Pure bigraphs: structure and dynamics. *Information and Computation*, 204(1):60–122, 2006.
- [114] Robin Milner. *The space and motion of communicating agents*. To appear. Cambridge University Press, 2009.
- [115] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100(1):1–40 and 41–77, 1992.
- [116] R. P. Nederpelt. *Strong normalization in a typed lambda-calculus with lambda-structured types*. PhD thesis, Eindhoven University of Technology, 1973.
- [117] M.H.A. Newman. On theories with a combinatorial definition of ‘equivalence’. *Annals of Mathematics*, 43(2):223–243, 1942.
- [118] Shane Ó Conchúir. Kind bigraphs – static theory. Technical Report TCD-CS-2005-36, Trinity College Dublin, May 2005.
- [119] Shane Ó Conchúir. Kind bigraphs. To appear. In *Proceedings of the Fourth Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT'06)*. Elsevier, 2006.
- [120] Shane Ó Conchúir. Proving PSN after ruining a perfectly good calculus. Technical Report TCD-CS-2006-49, Trinity College Dublin, September 2006.
- [121] Shane Ó Conchúir. Proving PSN by simulating non-local substitution with local substitution. In *Proceedings of The Third International Workshop on Higher-Order Rewriting (HOR 2006)*, August 2006.
- [122] Shane Ó Conchúir.  $\Lambda_{\text{sub}}$  as an explicit substitution calculus. Technical Report TR-2006-95, IT-Universitetet, København, Rued Langgards Vej 7, DK-2300 Copenhagen V, September 2006.



- [123] Shane Ó Conchúir. A bigraphical model of the simply typed  $\lambda$ -calculus. Technical Report TCD-CS-2007-38, Trinity College Dublin, July 2007.
- [124] C.-H. Luke Ong. Fully abstract models of the lazy lambda calculus. In *29th Annual Symposium on Foundations of Computer Science (FOCS), 24-26 October 1988, White Plains, New York, USA*, pages 368–376. IEEE, 1988.
- [125] David Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK, 1981. Springer-Verlag.
- [126] Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science (LICS '98)*. IEEE, Computer Society Press, June 1998.
- [127] Oren Patashnik. BibTeXing, 1988.
- [128] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.
- [129] Garrel Pottinger. A type assignment for the strongly normalizable  $\lambda$ -terms. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, London, 1980.
- [130] Eike Ritter. Characterising explicit substitutions which preserve termination. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference (TLCA'99), L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 325–339. Springer, 1999.
- [131] Kristoffer Høgsbro Rose. Explicit cyclic substitution. Semantics Note d-166, DIKU, University of Copenhagen, Denmark, March 1993.
- [132] Kristoffer Høgsbro Rose. Explicit substitution – tutorial & survey, 1996.
- [133] Kristoffer Høgsbro Rose. *Operational Reduction Models for Functional Programming Languages*. PhD thesis, DIKU, University of Copenhagen, Denmark, Universitetsparken 1, DK-2100 København Ø, February 1996. Available as DIKU report 96/1.
- [134] Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, 1973.

- [135] Davide Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1):120–153, 1994.
- [136] Davide Sangiorgi and David Walker. *The  $\pi$ -Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.
- [137] Vladimiro Sassone and Pawel Sobociński. Deriving bisimulation congruences using 2 - categories. *Nordic Journal of Computing*, 10(2):163–183, 2003.
- [138] Vladimiro Sassone and Pawel Sobociński. Reactive systems over cospans. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 311–320. IEEE Computer Society, 2005.
- [139] Peter Sewell. Global/local subtyping and capability inference for a distributed  $\pi$ -calculus. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium (ICALP'98), Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 695–706. Springer, 1998.
- [140] Peter Sewell. From rewrite rules to bisimulation congruences. *Theoretical Computer Science*, 274(1-2):183–230, 2002.
- [141] William W. Tait. Intensional interpretations of functionals of finite type i. *Journal of Symbolic Logic*, 32(2):198–212, 1967.
- [142] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.
- [143] Jerzy Tiuryn and Mitchell Wand. Untyped lambda-calculus with input-output. In Hélène Kirchner, editor, *Trees in Algebra and Programming: CAAP'96, Proc. 21st International Colloquium*, volume 1059, pages 317–329, Berlin, Heidelberg, and New York, 1996. Springer-Verlag.
- [144] David N. Turner. *The Polymorphic Pi-calculus: Theory and Implementation*. PhD thesis, University of Edinburgh, 1996.
- [145] Jan Vitek and Giuseppe Castagna. Seal: A framework for secure mobile computations. In Henri E. Bal, Boumediene Belkhouche, and Luca Cardelli, editors, *ICCL Workshop: Internet Programming Languages*, volume 1686 of *Lecture Notes in Computer Science*, pages 47–77. Springer, 1998.

- 
- [146] John Whitehouse. Sudoku solver. <http://www.eddaardvark.co.uk/sudokusolver.html>.
- [147] Pawel Wojciechowski and Peter Sewell. Nomadic pict: Language and infrastructure design for mobile agents. In *First International Symposium on Agent Systems and Applications (ASA '99)/Third International Symposium on Mobile Agents (MA '99)*, Palm Springs, CA, USA, 1999.
- [148] World Wide Web Consortium (W3C), <http://www.w3.org/XML/>. *Extensible Markup Language (XML)*.

# Appendices

## Appendix for Part I

4'33" – John Cage

In this appendix, we repeat some of the definitions for pure bigraphs for the convenience of the reader, demonstrate that Milner's homomorphic sorting is subsumed by kind sorting, vanquish past self-doubts on the applicability of kind sorting, and present the proofs that the dynamic theory is retained in kind sortings, certain subcategories of kind sortings, and our link sortings.

### A.1 Pure bigraphs

This section reproduces original definitions referred to in the dissertation. Most definitions are taken either verbatim or with minor modifications from the original sources [74, 110] to which we refer the reader for detailed explanations.

**Definition A.1** (consistency conditions for place graphs). *Three consistency conditions are defined on a pair  $\vec{A} : h \rightarrow \vec{m}$  of place graphs. Let  $i$  range over  $\{0, 1\}$ ; also let  $w_2, w'_2$  range over  $h \uplus V_2$ , the shared places.*

$$CP0 \quad ctrl_0(v_2) = ctrl_1(v_2)$$

$$CP1 \quad \text{If } A_i(w) \in V_2 \text{ then } w \in h \uplus V_2 \text{ and } A_{\bar{i}}(w) = A_i(w)$$

$$CP2 \quad \text{If } A_i(w_2) \in V_i - V_2 \text{ then } A_{\bar{i}}(w_2) \in m_{\bar{i}}, \text{ and if also } A_{\bar{i}}(w) = A_{\bar{i}}(w_2) \\ \text{then } w \in h \uplus V_2 \text{ and } A_i(w) = A_i(w_2).$$

**Definition A.2** (consistency conditions for link graphs). *Three consistency conditions are defined on a pair  $\vec{A} : W \rightarrow \vec{X}$  of link graphs. Let  $p$  range over arbitrary points,  $p_i, p'_i, \dots$  to range over  $P_i$ , and  $p_2, p'_2, \dots$  to range over  $W \uplus P_2$ , the shared points.*

$$CL0 \quad ctrl_0(v_2) = ctrl_1(v_2)$$

$$CL1 \quad \text{If } A_i(p) \in E_2 \text{ then } p \in W \uplus P_2 \text{ and } A_i(p) = A_i(p)$$

$$CL2 \quad \text{If } A_i(p_2) \in E_i - E_2 \text{ then } A_i(p_2) \in X_i, \text{ and if also } A_i(p) = A_i(p_2) \\ \text{then } p \in W \uplus P_2 \text{ and } A_i(p) = A_i(p_2).$$

A bigraph is called *lean* if its link graph is lean *i.e.* has no idle edges.  $A^E$  denotes the result of adding a set  $E$  of fresh idle edges to  $A$ .

**Definition A.3** (abstract pure bigraphs and their category). *Two concrete bigraphs  $A$  and  $B$  are lean-support equivalent, written  $A \approx B$ , if after discarding any idle edges they are support equivalent. The category  $\mathbf{BIG}(\mathcal{K})$  of abstract pure bigraphs has the same objects as  $\mathbf{BIG}(\mathcal{K})$ , and its arrows are lean-support equivalence classes of concrete bigraphs. Lean-support equivalence is a congruence (Definition 3.5). The associated quotient functor, assured by Definition 3.6, is*

$$\llbracket \cdot \rrbracket : \mathbf{BIG}(\mathcal{K}) \rightarrow \mathbf{BIG}(\mathcal{K}).$$

*The definition of  $\mathbf{BIG}_h(\mathcal{K})$  is analogous, with the restriction of  $\llbracket \cdot \rrbracket$  to  $\mathbf{BIG}_h(\mathcal{K})$  as quotient functor.*

**Definition A.4** (parallel product). *The parallel product of two bigraphs is defined on interfaces by  $\langle m, X \rangle \parallel \langle n, Y \rangle \stackrel{\text{def}}{=} \langle m + n, X \cup Y \rangle$ , and on bigraphs by*

$$G_0 \parallel G_1 \stackrel{\text{def}}{=} \langle G_0^P \otimes G_1^P, G_0^L \mid G_1^L \rangle : I_0 \otimes I_1 \rightarrow J_0 \parallel J_1$$

*when the interfaces exist and the node sets are disjoint<sup>1</sup>.*

**Definition A.5** (parallel product (alternate definition)). *Let  $G_0 \parallel G_1$  be defined. Then*

$$G_0 \parallel G_1 = \sigma(G_0 \otimes \tau G_1),$$

*where the substitutions  $\sigma$  and  $\tau$  are defined as follows: If  $z_i$  ( $i \in n$ ) are the names shared between  $G_0$  and  $G_1$ , and  $w_i$  are fresh names in bijection with the  $z_i$ , then  $\tau(z_i) = w_i$  and  $\sigma(w_i) = \sigma(z_i) = z_i$  ( $i \in n$ ).*

**Definition A.6** (prime product). *The prime product of two interfaces is given by*

$$\langle m, X \rangle \parallel \langle n, Y \rangle \stackrel{\text{def}}{=} \langle 1, X \cup Y \rangle.$$

<sup>1</sup>Note that the edge sets are not required to be disjoint. This is because the parallel product on link graphs unions the link maps of both link graphs, not requiring the edge sets to be disjoint.

For two bigraphs  $\vec{P} : \vec{I} \rightarrow \vec{J}$ , if  $I_0 \otimes I_1$  defined and  $n$  is the sum of the widths of  $J_0$  and  $J_1$ , we define their prime product by

$$P_0 \mid P_1 \stackrel{\text{def}}{=} \text{merge}_n \circ (P_0 \parallel P_1) : I_0 \otimes I_1 \rightarrow J_0 \otimes J_1.$$

**Definition A.7** (underlying discrete bigraph). *Every bigraph  $G$  in  $\mathcal{BIG}(\mathcal{K})$  or  $\mathcal{BIG}_h(\mathcal{K})$  can be expressed uniquely (up to iso) as  $G = (\omega \otimes \text{id}_n) \circ D$ , where  $\omega$  is a wiring and  $D$  is discrete.*

## A.2 Homomorphic sortings

A homomorphic sorting was successfully used to model finite CCS [113]. In this appendix, we describe homomorphic sortings as kind sortings and demonstrate that they cannot be used to properly encode simply typed  $\Lambda_{\text{sub}}$  using our signature (Definition 11.22). We must emphasise that this is not surprising – the sorting was introduced and used to model finite CCS which has a simple sorted grammar and introducing a stronger sorting would have been unnecessary – but we believe that this section may serve to better explain kind sorting.

This appendix should be read after Section 11.4.

### A.2.1 Homomorphic sortings as kind sortings

A homomorphic sorting is a place-sorting. It has the property that the children of a root or node all have the same sort and, further, a root has the same sort as all of its children. We present the definition slightly differently – and worse – here to fit our previous definitions.

**Definition A.8** (homomorphic signature). *A homomorphic signature  $\{\mathcal{K}, \Theta, \text{arity}, \text{actv}, \text{kind}, \text{sort}, \text{prnt}\}$  is composed of a set  $\mathcal{K}$  of controls, a set  $\Theta$  of sorts, and five maps:*

$$\begin{aligned} \text{arity} & : \mathcal{K} \rightarrow \mathbb{N} & \text{sort} & : \mathcal{K} \rightarrow \Theta \\ \text{actv} & : \mathcal{K} \rightarrow \{\text{passive}, \text{active}\} & \text{prnt} & : \Theta \rightarrow \Theta \\ \text{kind} & : \mathcal{K} \rightarrow \{\text{atomic}, \text{non-atomic}\}. \end{aligned}$$

where atomic controls must be passive. Without loss of generality, we will assume  $\text{sort}$  is surjective (a partition). We denote an arbitrary homomorphic sorting as  $(\mathcal{K}, \Theta)$  with assumed functions  $\text{arity}, \text{actv}, \text{kind}, \text{sort}$ , and  $\text{prnt}$ .

**Definition A.9** (homomorphic grouping). *Let  $(\mathcal{K}, \Theta)$  be a homomorphic signature. For each control  $K \in \mathcal{K}$ , the set  $(\text{sort}^{-1} \circ \text{sort})(K)$ , the homomorphic grouping of  $K$ , is denoted by  $\Theta_K$ .*

If  $K$  and  $L$  have the same sort then  $\Theta_K = \Theta_L$ .

**Definition A.10** (homomorphic sorting [113]). A place-sorting  $\Sigma_K = (\mathcal{K}, \Theta, \Phi)$  over a homomorphic signature  $(\mathcal{K}, \Theta)$  is a homomorphic sorting if for each site or node  $w$  in a bigraph  $G$ :

- if  $G(w) = v$  then  $\text{sort}(v) = \text{prnt}(\text{sort}(w))$ ;
- if  $G(w) = r$  then  $\text{sort}(r) = \text{sort}(w)$ .

where  $v$  is a node and  $r$  is a root.

Homomorphic sorting can be used to encode finite CCS in bigraphs [113]. It may be described as a kind sorting. Figure A.1 may help to illustrate this by summarising the main conditions on each sorting with respect to nodes (*parent* denotes the parent map of the place graph). Note that for each non-atomic  $K \in \mathcal{K}$ , the set  $(\text{sort}^{-1} \circ \text{prnt}^{-1} \circ \text{sort})(K)$  is the set of controls which nodes of  $K$  may be a parent of according to the homomorphic sorting.

**Definition A.11** (homomorphic kind signature). Let  $\{\mathcal{K}, \Theta, \text{arity}, \text{actv}, \text{kind}, \text{sort}, \text{prnt}\}$  be a homomorphic signature. The corresponding homomorphic kind signature  $\{\mathcal{K}, \text{arity}_h, \text{actv}_h, \text{kind}_h\}$  is the fundamental kind signature where  $\text{arity}_h = \text{arity}$ ,  $\text{actv}_h = \text{actv}$ , and

$$\begin{aligned} \text{kind}_h(K) &= (\text{sort}^{-1} \circ \text{prnt}^{-1} \circ \text{sort})(K) && \text{if } \text{kind}(K) = \text{non-atomic}, \\ \text{kind}_h(K) &= \emptyset && \text{if } \text{kind}(K) = \text{atomic}. \end{aligned}$$

The associated kind sorting satisfies most of the conditions of a homomorphic sorting. The final condition is that the roots are sorted. For this, we use the homomorphic s-category associated with the sorting (where the interface sorts are the homomorphic groupings of the signature).

### A.2.2 Homomorphic sorting and simply typed $\Lambda_{\text{sub}}$

One property we want of our model is that typing is somehow preserved (Proposition 11.34). We cannot achieve this with homomorphic sorting.

$$\begin{array}{ccc} \mathcal{K} \xrightarrow{\text{srt}} \Theta & \begin{array}{c} \text{prnt} \\ \curvearrowright \end{array} & \mathcal{K} \xrightarrow{\text{knd}} \mathcal{P}(\mathcal{K}) \\ & & \begin{array}{c} \subseteq \\ \curvearrowright \end{array} \end{array}$$

$$\text{srt}(\text{parent}(v)) = \text{prnt}(\text{sort}(v)) \qquad \{v' \mid \text{parent}(v') = v\} \subseteq \text{knd}(v)$$

**Figure A.1:** Containment conditions of homomorphic sortings and kind sortings



**Example A.12** ( $\mathcal{K}^\rightarrow$  cannot be sorted homomorphically for our purposes). We will try to homomorphically sort  $\mathcal{K}^\rightarrow$ . Give  $\mathbf{var}^G$  the sort  $\theta_1$  for some  $A$ . In our model,  $\mathbf{var}^G$  should be able to be a child of both  $2^G$  and  $\mathbf{def}^G$ . According to the definition of homomorphic sorting, both  $2^G$  and  $\mathbf{def}^G$  must have the same sort  $\theta_2$ . Let  $\mathbf{app}^{G \rightarrow G, G}$  have the sort  $\theta_3$ . In our model,  $\mathbf{app}^{G \rightarrow G, G}$  should be able to contain  $2^G$ . However, as  $2^G$  and  $\mathbf{def}^G$  have the same sort, the homomorphic sorting allows  $\mathbf{def}^G$  to be a child of  $\mathbf{app}^{G \rightarrow G, G}$ . This breaks the sorting of the controls (Definition 11.24) leading to bigraphs which do not represent simply typed  $\Lambda_{\text{sub}}$  terms.

The problem gets worse. Let  $G' \neq G$ . In our model,  $\mathbf{var}^G$  should be able to be a child of  $\mathbf{lam}^{G' \rightarrow G}$  as well. Therefore, the sort of  $\mathbf{lam}^{G' \rightarrow G}$  is also  $\theta_2$ . This means that using the homomorphic sorting allows  $\mathbf{lam}^{G' \rightarrow G}$  to be a child of  $\mathbf{app}^{G \rightarrow G, G}$  which does not respect the typing of simply typed  $\Lambda_{\text{sub}}$ .

The problem with using homomorphic sorting for our model is that the sorts are merged – degeneracy creeps in. In the example above, the function  $\mathit{prnt} : \Theta \rightarrow \Theta$  is (partially) defined as

$$\mathit{prnt}(\theta_1) \mapsto \theta_2, \quad \mathit{prnt}(\theta_2) \mapsto \theta_3, \quad \mathit{prnt}(\theta_3) \mapsto \theta_2.$$

This looping structure ( $\theta_2 \mapsto \theta_3 \mapsto \theta_2$ ) comes from  $\mathit{prnt}$  being an endomap (see Lawvere and Schanuel’s book [89] for some graphical intuitions). Endomaps can be seen as a special sort of restricted directed graph. Kind sortings are essentially relations and relations may be described by directed graphs. This intuition fits the fact that homomorphic sortings are special cases of kind sortings [118]. Put another way, homomorphic sortings are based on functions whereas kind sortings are based on relations.

### A.2.3 Applicability of kind sortings

Previously [118], we questioned the applicability of kind sortings based on the facts that: i) kind sortings do not reflect pushouts in general; and ii) fitting subcategories do reflect pushouts but cannot describe all pure Brss. In this section, we address these doubts based on what we have learned from exploring subcategories of kind sortings.

Subsequently, Bundgaard and Sassone answered the first problem by showing that weak pushout reflection sufficed to retain the adequacy theorem of Leifer and Milner; we have proven that jointly opcartesian bounds (and hence IPOs) of kind sorted s-categories reflect pushouts.

This observation renders the second problem irrelevant but two questions are raised: i) what was wrong with our example that fitting subcategories cannot describe all pure Brss? and ii) what is the proper generalisation from pure Brss to kind Brss?

The first question has been answered by Milner. The counter-example we gave for fitting subcategories was based on a reaction rule for the  $\pi$ -calculus with summation given by Jensen and Milner [74] and reproduced in Figure A.2. There is no sorting for the outer interface which is fitting for both redex and reactum (which must have the same sort) in the fitting subcategory of such a model. The key mistake here was our choice of subcategory; a fitting subcategory was merely the wrong choice. The grammar of the  $\pi$ -calculus with summation (and replication) [136] is given by

$$\begin{aligned} P &:= M \mid P \mid P \mid \nu z P \mid ! P \\ M &:= \mathbf{0} \mid \pi.P \mid M + M. \end{aligned}$$

where  $\pi$  is a *prefix* (typically input and output). A better choice would be a two-sorted homomorphic subcategory like Milner used for finite CCS. Indeed, Jensen chooses such a subcategory to sort  $\pi$ -calculus with summation [73]. The outer interface of the rule would then be sorted with the ‘process’ sort.

As for the question of whether fundamental kind Brss generalise pure Brss properly, this can be answered by taking the full subcategory over the lifted signature (define  $\text{kind}(K) = \mathcal{K}$  for non-atomic controls,  $\text{kind}(K) = \emptyset$  for atomic controls) where the only allowed interface sort is  $\mathcal{K}$  *i.e.* each place can contain nodes of all controls. Let  $\mathcal{I} : \mathbf{C} \rightarrow \mathbf{BIG}(\Sigma_K)$  be such a subcategory of a kind sorting  $\mathcal{U} : \mathbf{BIG}(\Sigma_K) \rightarrow \mathbf{BIG}(\mathcal{K})$ . The functor  $\mathcal{UI}$  is an isomorphism of s-categories; the objects of  $\mathbf{C}$  are in bijection with the pure interfaces of  $\mathbf{E}$  and, since the kind rules are vacuously true, so are the homsets.

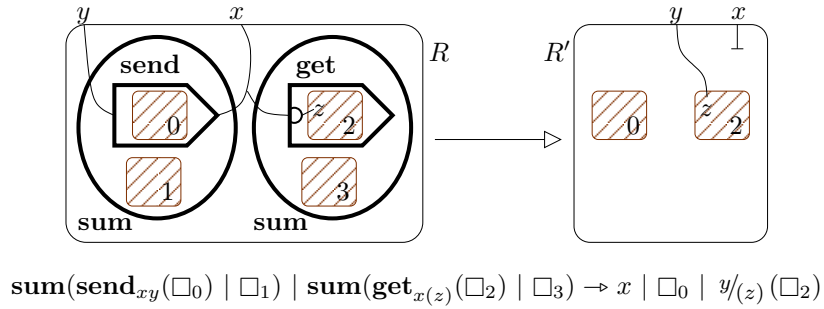


**Figure A.2:** Reaction rule for the  $\pi$ -calculus with summation [74]

### A.3 Proofs for Section 5.2

**Lemma A.13.** *A cospan  $\vec{A}: \vec{H} \rightarrow I$  of a kind sorted  $s$ -category is jointly opcartesian with respect to the sorting functor iff the sort of each place of  $I$  has the least outer sort which satisfies the sorting conditions for  $\vec{A}$ .*

*Proof.*

$\Rightarrow$  Assume a place  $r$  of  $I$  does not have the least outer sort for the cospan. Let  $\vec{B}: \vec{H} \rightarrow I'$  have the same place graph and link graphs as  $\vec{A}$  where  $I = I'$  except that  $r$  has the least outer sort for the cospan in  $I'$ . Any mediating arrow from  $\vec{A}$  to  $\vec{B}$  must be a vertical. Hence, there is a mediating arrow between  $\vec{A}^u$  and  $\vec{B}^u$ . However since  $r$  has a lesser sort in  $I'$  than  $I$ , any mediating arrow would break the condition K2.

$\Leftarrow$  Assume all places of  $I$  have the least outer sort which satisfies the sorting conditions for  $\vec{A}$ . Let  $\vec{B}: \vec{H} \rightarrow J$  be a second cospan such that  $F \circ A_i^u = B_i^u$  for some pure bigraph  $F$ . As  $\mathcal{U}$  is faithful, it suffices to define an arrow  $F': I \rightarrow J$  with the same place graph and link graph as  $F$  (so that  $\mathcal{U}(F') = F$ ) and prove this arrow satisfies the sorting.

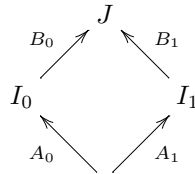
**K1** Let  $p = F'(v)$ . As  $F \circ A_i^u = B_i^u$  and  $B_i$  is sorted, we have  $ctrl(v) \in kind(p)$ .

**K2** Let  $p = F'(r)$ . As  $r$  has the least outer sort which satisfies the sorting conditions for  $\vec{A}$ , if  $K \in kind(r)$  then either  $v <_{A_i} r, ctrl(v) = K$  or  $s <_{A_i} r, K \in kind(s)$ . As  $F \circ A_i^u = B_i^u$ , this implies that  $v <_{B_i} p$  or  $s <_{B_i} p$ . Therefore, by K1 and K2 of  $B_i$ ,  $K \in kind(p)$ .

**K3** As  $F'$  has the same parent graph as  $F$ , atomic nodes in  $F'$  have no children.

□

**Proposition A.14** (opcartesian redexes imply opcartesian labels in JOT of a kind sorted Brs). *Let  $(B_0, B_1)$  be a jointly opcartesian bound for  $(A_0, A_1)$  and  $A_1$  be opcartesian. Then  $B_0$  is opcartesian.*



*Proof.* We know that  $J$  is fitting for the pair  $(B_0, B_1)$  and that  $B_0 \circ A_0 = B_1 \circ A_1$ . In the following,  $t$  is a root of  $B_0$  and  $B_1$ ,  $r_i$  is a site of  $B_i$ , and  $s$  is a site of  $A_0$  and  $A_1$ .  $v$  ranges over nodes. We will show that if  $K \in \theta$  where  $\text{sort}(t) = \theta$  then  $K \in \theta$  is necessary for K1 and K2 to hold in  $B_0$ .

Let  $K \in \theta$  where  $\text{sort}(t) = \theta$ . We will assume that  $K \in \theta$  is not necessary for K1 and K2 to hold in  $B_0$ . Since  $J$  is fitting for  $\vec{B}$ , it must then be that  $K \in \theta$  is necessary for K1 or K2 to hold in  $B_1$ . We have a case split:

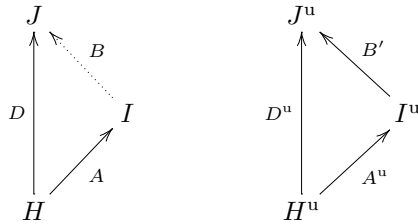
- Let  $t = B_1(v)$  and  $\text{ctrl}(v) = K$ . If  $v \in B_0$  then  $t = B_0(v)$  which contradicts our assumption. Otherwise, we have  $r_0 = A_0(v)$  and  $t = B_0(r_0)$ . But by K1 of  $A$ ,  $K \in \text{sort}(r_0)$  and we reach a contradiction again.
- Let  $t = B_1(r_1)$  and  $K \in \text{sort}(r_1)$ . As  $A_1$  is fitting,  $K \in \text{sort}(r_1)$  is necessary for K1 or K2 to hold in  $A_1$ . We have a further case split:
  - Say  $K \in \text{sort}(r_1)$  is necessary for K1 to hold in  $A_1$ . Then  $r_1 = A_1(v)$  and  $\text{ctrl}(v) = K$ . Therefore, either  $t = B_0(v)$  and so our assumption is contradicted by K1 or else  $t = B_0(r_0)$  with  $r_0 = A_0(v)$  in which case by K1 of  $A_0$ ,  $K \in \text{sort}(r_0)$  and K2 of  $B_0$  contradicts our assumption.
  - Say  $K \in \text{sort}(r_1)$  is necessary for K2 to hold in  $A_1$ . Then  $r_1 = A_1(s)$  and  $K \in \text{sort}(s)$ . But then  $t = B_0(r_0)$  and  $r_0 = A_0(s)$ . K2 of  $A_0$  implies  $K \in \text{sort}(r_0)$  and so K2 of  $B_0$  contradicts our assumption.

Thus, for all  $t$  in  $J$  and all  $K \in \text{sort}(t)$ ,  $K \in \text{sort}(t)$  is necessary for K1 and K2 to hold in  $B_0$ . Hence,  $B_0$  is a fitting bigraph.  $\square$

The definition of pure functor of a subcategory of a sorting was given on page 4.21.

**Lemma A.15.** *All arrows of fitting meet subcategories of kind sortings are opcartesian with respect to their pure functor.*

*Proof.*



Let  $A : H \rightarrow I$  be an arrow of a fitting meet subcategory where  $D : H \rightarrow J$  and where  $B' : I^u \rightarrow J^u$  is a pure bigraph such that  $D^u = B' \circ A^u$ . Let  $B : I \rightarrow J$  have the same *prnt* map as  $B'$ . We show that  $B$  is a fitting arrow of  $\mathcal{BIG}(\Sigma_K)$  *i.e.* that it obeys the kind sorting conditions.

We start by proving that  $B$  obeys the sorting. In the following,  $v$  and  $v'$  are nodes,  $r$  is a site of  $B$ ,  $s$  is a site of  $A$ , and  $p$  is a root or node of  $B$ .

**K1** Let  $p = B(v)$ . Then  $p = D(v)$  and since  $D$  is sorted,  $B$  obeys the sorting.

**K2** Let  $p = B(r)$ . We must prove  $kind(r) \subseteq kind(p)$ .

As  $A$  is sorted, the sort of  $r$  is defined as

$$kind(r) = \mathcal{K}_r \uplus \left( \bigcup_{v <_A r} ctrl(v) \right) \cup \left( \bigcup_{s <_A r} kind(s) \right)$$

where by the definition of fitting s-category,  $\mathcal{K}_r$  is the smallest possible set. As  $D^u = B' \circ A^u$ , we have  $v <_A r$  implies  $v <_D p$  and  $s <_A r$  implies  $s <_D p$ . As  $D$  is sorted, the sort of  $p$  can be therefore be defined as

$$kind(p) = \mathcal{K}_p \uplus \left( \bigcup_{v <_A r} ctrl(v) \right) \cup \left( \bigcup_{s <_A r} kind(s) \right).$$

As  $\mathcal{BIG}(\Sigma_K)$  is a meet s-category, the interface sort

$$kind(r) \cap kind(p) = \mathcal{K}' \uplus \left( \bigcup_{v <_A r} ctrl(v) \right) \cup \left( \bigcup_{s <_A r} kind(s) \right)$$

is defined. This sort can sort the root  $r$  in  $A$  and as  $kind(r) \cap kind(p) \subseteq kind(r)$ , and  $kind(r)$  is the smallest sort to sort  $r$ , it must be that  $kind(r) = kind(r) \cap kind(p)$ . Thus,  $kind(r) = kind(r) \cap kind(p) \subseteq kind(p)$ .

**K3** Trivially follows from  $D^u = B' \circ A^u$  as atomic controls map to atomic controls under  $\mathcal{U}$ .

We now prove  $B$  is fitting. Let  $t$  be a place in  $J$ . As  $D$  is fitting,  $t$  has the sort

$$kind(t) = \mathcal{K}_t \uplus \left( \bigcup_{v <_D t} ctrl(v) \right) \cup \left( \bigcup_{s <_D t} kind(s) \right).$$

where  $\mathcal{K}_t$  is minimal. As  $B$  is sorted,  $t$  must at least have the sort

$$\begin{aligned} kind(t) &= \bigcup_{v <_B t} ctrl(v) \cup \bigcup_{r <_B t} kind(r) \\ &= \bigcup_{v <_B t} ctrl(v) \cup \bigcup_{r <_B t} \left( \mathcal{K}_r \cup \bigcup_{v <_A r} ctrl(v) \cup \bigcup_{s <_A r} kind(s) \right) \\ &= \bigcup_{r <_B t} \mathcal{K}_r \cup \left( \bigcup_{v <_D t} ctrl(v) \right) \cup \left( \bigcup_{s <_D t} kind(s) \right). \end{aligned}$$

Therefore,

$$K_t = \bigcup_{r <_B t} K_r \setminus \left( \left( \bigcup_{v <_D t} \text{ctrl}(v) \right) \cup \left( \bigcup_{s <_D t} \text{kind}(s) \right) \right)$$

and so  $K_t$  is the set of controls which are unnecessary to sort  $t$  in  $B$ . If  $B$  was not fitting, then there would imply that  $K_t$  was not minimal which is a contradiction as  $D$  is fitting.  $\square$

**Lemma A.16.** *Let  $D = B \circ A$  and  $D$  and  $A$  be opcartesian arrows of a kind sorted  $s$ -category with respect to the sorting functor. Then  $B$  is opcartesian.*

*Proof.* We will prove that when  $K \in \text{sort}(t)$ , where  $t$  is a root of both  $B$  and  $D$ ,  $K \in \text{sort}(t)$  is required for K1 and K2 to hold in  $B$ .

Let  $K \in \text{sort}(t)$ . Since  $D$  is fitting, we know that this membership is necessary for K1 or K2 to hold in  $D$ . We examine the two cases.

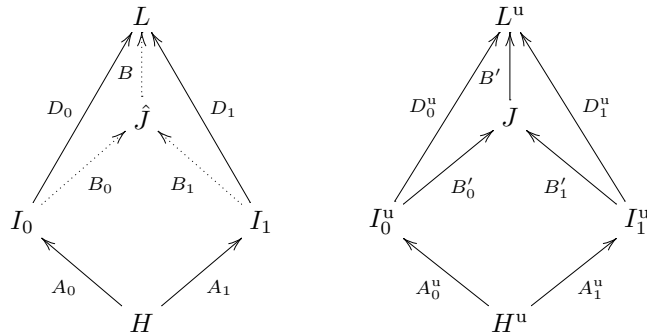
Let  $t = D(v)$  and  $K = \text{ctrl}(v)$ . The node  $v$  is located in either  $B$  or  $A$ . If  $v$  is in  $B$ , then  $t = B(v)$  and  $K \in \text{sort}(t)$  is necessary for K1 of  $B$ . If  $v$  is in  $A$ , then  $r = A(v)$  and  $t = B(r)$ . By K1 of  $A$ ,  $K \in \text{sort}(r)$  and by K2 of  $B$  we require  $K \in \text{sort}(t)$ .

Let  $t = D(s)$  and  $K \in \text{sort}(s)$ . Then  $r = A(s)$  and  $t = B(r)$ . By K2 of  $A$ ,  $K \in \text{sort}(r)$ . By K2 of  $B$ , we require  $K \in \text{sort}(t)$ .  $\square$

## A.4 Proofs for Section 5.3

**Construction A.17** (building a kind RPO).

*Proof.* Let  $\vec{A} : H \rightarrow \vec{I}$  have a bound  $\vec{D} : \vec{I} \rightarrow L$  in  $\mathbf{BIG}(\Sigma_K)$ . We wish to build a kind RPO  $(\vec{B} : \vec{I} \rightarrow J, B : J \rightarrow L)$ . We start by building a pure RPO  $(\vec{B}', B')$  with interface  $J$  for  $\vec{A}^u$  to  $\vec{D}^u$ . From this we shall construct a kind bound  $(\vec{B}, B)$  with interface  $\hat{J}$  for  $\vec{A}$  to  $\vec{D}$ , such that  $(\vec{B}, B)^u = (\vec{B}', B')$ , by defining a kind interface for the pure RPO and then proving the triple  $(\vec{B}, B)$  sorted.



From Definition 4.6, we define the sort of  $r \in \hat{J}$  as:

$$\theta = \bigcup_{v <_{B_i} r, i \in \{0,1\}} ctrl(v) \cup \bigcup_{s <_{B_i} r, i \in \{0,1\}} kind(s).$$

By the definition of  $kind(r)$  and Lemma A.13, we need only prove that the pair  $\vec{B}$  is sorted. We treat the case of  $B_0$ .

**K1** Let  $p = B_0(v)$ . If  $p$  is a node then the proof follows from K1 of  $D_0$  and as  $B' \circ B'_0 = D_0$ .

If  $p$  is a root then by the definition above,  $ctrl(v) \in kind(p)$ .

**K2** Let  $p = B_0(s)$ . If  $p$  is a node then the proof follows from K2 of  $D_0$  and as  $B' \circ B'_0 = D_0$ .

If  $p$  is a root then by the definition above,  $kind(s) \in kind(p)$ .

**K3** As  $B_0$  has the same parent graph as  $B'_0$ , atomic nodes in  $B_0$  have no children.

Hence,  $(\vec{B}, B)$  is a kind bound for  $\vec{A}$  to  $\vec{D}$ . □

**Proposition A.18** (kind RPOs). *A kind RPO for  $\vec{A}$  to  $\vec{D}$  is provided by Construction A.17.*

*Proof.* Let  $(\vec{B}, B)$  be as in the construction. Let  $(\vec{C}, C)$  be any other bound for  $\vec{A}$  relative to  $\vec{D}$ . We must find a unique mediating arrow  $F$  from  $\vec{B}$  to  $\vec{C}$ .

From the construction,  $(\vec{B}, B)^u$  is a pure RPO for  $\vec{A}^u$  to  $\vec{D}^u$ .  $(\vec{C}, C)^u$  is also a bound for  $\vec{A}^u$  relative to  $\vec{D}^u$  and so there is a unique mediating arrow  $F'$  from  $\vec{B}^u$  to  $\vec{C}^u$ . By Lemma A.13, there is a unique mediating arrow  $F$  from  $\vec{B}$  to  $\vec{C}$  (furthermore,  $\mathcal{U}(F) = F'$ ). □

**Proposition A.19.** *If  $\mathcal{F} : \mathbf{A} \rightarrow \mathbf{B}$  and  $\mathcal{G} : \mathbf{B} \rightarrow \mathbf{C}$  create RPOs then  $\mathcal{GF}$  creates RPOs.*

*Proof.* Let  $(D_0, D_1)$  be a bound for  $(A_0, A_1)$  in  $\mathbf{A}$  and  $(B_0, B_1, B)$  be an RPO for  $(\mathcal{GF}(D_0), \mathcal{GF}(D_1))$  relative to  $(\mathcal{GF}(A_0), \mathcal{GF}(A_1))$  in  $\mathbf{C}$ .

$(\mathcal{F}(D_0), \mathcal{F}(D_1))$  is a bound for  $(\mathcal{F}(A_0), \mathcal{F}(A_1))$ . As  $\mathcal{G}$  creates RPOs, there exists an RPO  $(B'_0, B'_1, B')$  for  $(\mathcal{F}(D_0), \mathcal{F}(D_1))$  relative to  $(\mathcal{F}(A_0), \mathcal{F}(A_1))$  in  $\mathbf{B}$ . As  $\mathcal{F}$  creates RPOs, there exists an RPO  $(B''_0, B''_1, B'')$  for  $(D_0, D_1)$  relative to  $(A_0, A_1)$ . Therefore,  $\mathcal{GF}$  creates RPOs. □

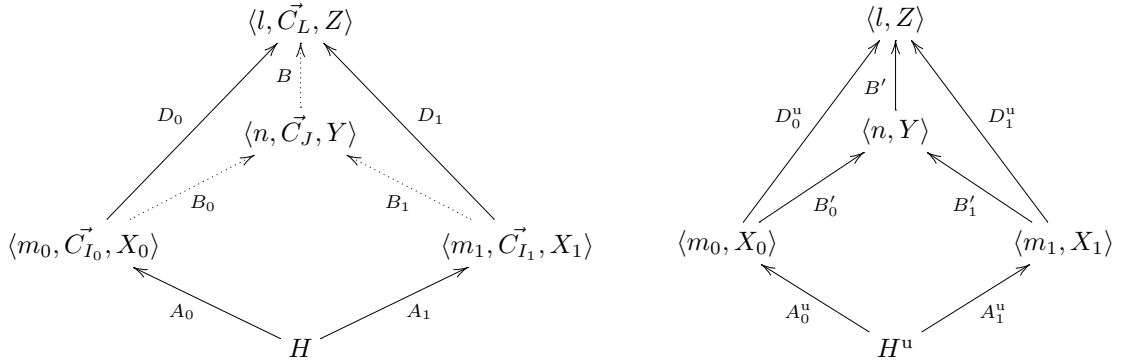
**Construction A.20** (building RPOs). *Let  $\vec{A} : H \rightarrow \vec{I}$  have a bound  $\vec{D} : \vec{I} \rightarrow L$  in a subcategory  $\mathbf{A}$  of a kind  $s$ -category  $\mathbf{BIG}(\Sigma_K)$ . We define an RPO  $\vec{B} : \vec{I} \rightarrow \hat{I}, B : \hat{I} \rightarrow L$  for this square as follows when:*

1.  $\mathbf{A}$  is a downward closed and controlled fitting subcategory;
2.  $\mathbf{A}$  is a downward closed and unioned fitting subcategory;

3.  $\mathbf{A}$  is a unioned full subcategory;

4.  $\mathbf{A}$  is a partitioned subcategory.

*Proof.* The first steps for the different cases are the same. We start by building a pure RPO  $(\vec{B}', B')$  for  $\vec{A}^u$  to  $\vec{D}^u$  (Construction 2.38). From this we shall construct a kind sorted bound  $(\vec{B}, B)$  for  $\vec{A}$  to  $\vec{D}$ , such that  $(\vec{B}, B)^u = (\vec{B}', B')$ . Then in the next proposition we shall prove the universal property of this relative bound. The situation is depicted below.



For each case, we will define the sorts of places in the interface  $\hat{I}$  and then prove that triple  $(\vec{B}, B)$  is kind sorted and consists of bigraphs in the s-category we are working in. In order to define the sorts, we must revisit Jensen and Milner's construction and see how each place of  $\hat{I}$  arises in the construction.

Let  $\hat{r}$  be a place of  $\hat{I}$ .  $\hat{r}$  is not barren in either  $B'_0$  or  $B'_1$ <sup>2</sup>. Let  $r_i$  denote a site of  $B_i$ ,  $i \in \{0, 1\}$ . The controls of nodes of  $B'_0$  which are children of  $\hat{r}$  and the nodes of  $B'_1$  which are children of  $\hat{r}$  are respectively defined as

$$\begin{aligned} V_{B'_0} &= \bigcup_{r_1 <_{B'_1} \hat{r}} \left( \bigcup_{v <_{A_1} r_1} \text{ctrl}(v) \right), \\ V_{B'_1} &= \bigcup_{r_0 <_{B'_0} \hat{r}} \left( \bigcup_{v <_{A_0} r_0} \text{ctrl}(v) \right). \end{aligned}$$

In order that  $\hat{r}$  respects the kind rules for both  $B_0$  and  $B_1$ , we must have

$$\text{kind}(\hat{r}) = \mathcal{K}_r \uplus \left( (V_{B'_0} \cup V_{B'_1}) \cup \left( \bigcup_{r_0 <_{B'_0} \hat{r}} \text{kind}(r_0) \right) \cup \left( \bigcup_{r_1 <_{B'_1} \hat{r}} \text{kind}(r_1) \right) \right)$$

for some smallest set  $\mathcal{K}_r$  such that  $\text{kind}(\hat{r})$  is an interface sort. However, as  $B'_0 \circ A_0^u = B'_1 \circ A_1^u$ , if  $v_1 <_{B_0} \hat{r}$  then  $v_1 <_{A_1} r_1$  for some  $r_1 <_{B'_1} \hat{r}$ . As  $A_0$  and  $A_1$  are sorted, we can then rewrite the

<sup>2</sup>This is a familiar property in colimits of diagrams of sets and set-like categories.



equation above as

$$kind(\hat{r}) = \mathcal{K}_r \uplus \left( \left( \bigcup_{r_0 <_{B'_0} \hat{r}} kind(r_0) \right) \cup \left( \bigcup_{r_1 <_{B'_1} \hat{r}} kind(r_1) \right) \right). \quad (\text{A.1})$$

Before we break the remainder of the construction over the cases, we state the following sub-proposition:

If an interface which satisfies the equation (A.1) exists in the s-category this interface defines  $B_0$  and  $B_1$  as kind sorted bigraphs *i.e.* conditions K1-K3 are satisfied. (P1)

By the definition of  $kind(\hat{r})$ , K1 and K2 are satisfied on roots of  $\vec{B}$ . The commutativity of the diagram on the right above proves K1, K2, and K3 on nodes of  $\vec{B}$ . Similarly, commutativity of the diagram on the right and the fact that no place of  $\hat{I}$  is barren in  $B_0$  or  $B_1$  proves that K1 and K3 are satisfied in  $B$ .

We now explicitly define the sort of a place  $\hat{r}$  of  $\hat{I}$  for the separate cases. For the fitting subcategories, we must also show that this sort is the smallest sort satisfying (A.1) in order that  $B_0$  and  $B_1$  are arrows of that subcategory. We must also show that  $B$  satisfies K2.

1. For a downward closed and controlled fitting subcategory  $\mathbf{A}$ , we define

$$kind(\hat{r}) = \left( \bigcup_{r_0 <_{B'_0} \hat{r}} kind(r_0) \right) \cup \left( \bigcup_{r_1 <_{B'_1} \hat{r}} kind(r_1) \right). \quad (\text{A.2})$$

We first prove that this is an interface sort.

Let  $\hat{r} <_{B'} p$  where  $p$  is a root or node. For any  $r_i, r_i <_{B'_i} \hat{r}, i \in \{0, 1\}$ , we have  $kind(r_i) \subseteq kind(p)$  as  $D_i$  is sorted. Therefore,  $kind(\hat{r}) \subseteq kind(p)$ . As  $\mathbf{A}$  is downwards closed and each control sort is an interface sort,  $kind(\hat{r})$  is therefore an interface sort and does not break K2 in  $B$ . We must prove that it is also the smallest sort for  $B_0$  or  $B_1$  individually *i.e.* that  $B_0$  and  $B_1$  are arrows of  $\mathbf{A}$ .

Say  $K \in kind(r_1), r_1 <_{B_1} \hat{r}$ . We will prove that either  $v <_{B_0} \hat{r}, ctrl(v) = K$  or  $K \in kind(r_0), r_0 <_{B_0} \hat{r}$ . This suffices to prove that  $kind(\hat{r})$  is the smallest interface sort for  $B_0$ . Similar reasoning holds for  $B_1$ . As  $K \in kind(r_1)$  then either:

- (a)  $v <_{A_1} r_1, ctrl(v) = K$ .

If  $v$  is a shared node then, by the RPO construction, there exists a site  $r_0 \in m_0$  such that  $v <_{A_0} r_0$  and  $r_0 <_{B_0} \hat{r}$ . Therefore,  $K \in kind(r_0)$ .

If  $v$  is not shared then it is also a node of  $B_0$  such that  $v <_{B_0} \hat{r}$ .

- (b)  $s <_{A_1} r_1, K \in \text{kind}(s)$ .

This case is the same as for a shared node.

- (c)  $K$  is some other control unnecessary to sort  $A_1$ .

As all subsets of an interface sort are interface sorts, this case never occurs in this fitting s-category.

Therefore, the sort above is the smallest interface sort of the fitting s-category which sorts  $B_0$  and  $B_1$  and so these are arrows of the category. By Lemma A.15, so is  $B$ .

2. For a downward closed and unioned fitting subcategory  $\mathbf{A}$ , we define  $\text{kind}(\hat{r})$  as in equation (A.2) above. This is an interface sort by definition and – using reasoning similar to the last case – the smallest which sorts both  $B_0$  and  $B_1$ . The remainder of this case proceeds as in the last case.
3. For a unioned full subcategory  $\mathbf{A}$ , we define  $\text{kind}(\hat{r})$  as in equation (A.2) above. This is an interface sort by definition and – using the above argument again – the smallest which sorts both  $B_0$  and  $B_1$  (although not necessarily the smallest sort for  $B_0$  or  $B_1$  individually). We must now prove that  $B$  satisfies K2.

Let  $p$  be a node or root of  $B'$ . We must prove that if  $\hat{r} <_B p$  then  $\text{kind}(\hat{r}) \subseteq \text{kind}(p)$  for  $\hat{r} \in \hat{I}$ . Let  $K \in \text{kind}(\hat{r})$ . Then  $K \in \text{kind}(r_i)$  for some  $r_i \in m_i$  where  $r_i <_{B_i} \hat{r}$  and  $i \in \{0, 1\}$ . As  $r_i <_{D_i} p$  and  $D_i$  is sorted,  $K \in \text{kind}(p)$ . Therefore, any element of  $\text{kind}(\hat{r})$  is an element of  $\text{kind}(p)$  and  $B$  is sorted.

4. For a partitioned s-category, we begin by letting  $\mathcal{K}_r = \emptyset$  and proving that all places in equation (A.1) have the same sort.

From the pure RPO construction,  $\hat{r}$  must parent at least one site in either  $B_0$  or  $B_1$ .

If  $\hat{r}$  has no sites as children in  $B_1$  then it must have exactly one site  $r_0$  as a child in  $B_0$  and so  $\text{kind}(\hat{r}) = \text{kind}(r_0)$ . This sorts  $B_0$  as  $\hat{r}$  has no other children in  $B_0$ . The children of  $\hat{r}$  in  $B_1$  are the children of  $r_0$  in  $A_0$  and so  $B_1$  is also sorted.

Let  $\hat{r}$  have a non-zero number of sites as children in both  $B_0$  and  $B_1$ . If we examine the equivalence relation  $\cong$  over these sites defined in the pure construction, we can see that the relation that the equivalence relation is based on relates sites which both parent a shared node. In a partitioned s-category, this implies they have the same sort. Therefore, for any pair  $(r_0 \in m_0, r_1 \in m_1)$  such that  $r_0 <_{B_0} \hat{r}, r_1 <_{B_1} \hat{r}, \text{kind}(r_0) = \text{kind}(r_1)$ . Therefore,  $\text{kind}(\hat{r}) = \text{kind}(r_i)$  where  $r_i <_{B_i} \hat{r}, i \in \{0, 1\}$ . As above, this sorts  $B_0$  and  $B_1$ .

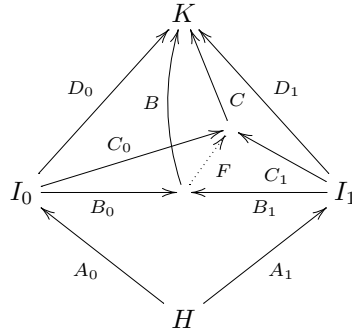
Finally, let  $p >_{B'} \hat{r}$ . Say  $\hat{r}$  is the parent of a place  $r_i \in m_i$  in  $B_i$ . Then  $p >_{D_i} r_i$  and since  $D_i$  is sorted,  $\text{kind}(\hat{r}) = \text{kind}(r_i) \subseteq p$ . Therefore,  $B$  is sorted.

□

We will now prove that the constructions above indeed build RPOs. A crucial point is that the interface sort of the RPO interface is constructed to have the minimum available interface sort of the subcategory.

**Proposition A.21** (creation of RPOs). *Whenever  $\vec{D}$  bounds  $\vec{A}$  in a subcategory  $\mathcal{I}$  of a kind sorting then any RPO for  $\mathcal{U}(\vec{A})$  relative to  $\mathcal{U}(\vec{D})$  has a unique  $\mathcal{U}$ -preimage that is an RPO for  $\vec{A}$  relative to  $\vec{D}$  if:*

1.  $\mathcal{I}$  is a downward closed and controlled fitting subcategory;
2.  $\mathcal{I}$  is a downward closed and unioned fitting subcategory;
3.  $\mathcal{I}$  is a unioned full subcategory;
4.  $\mathcal{I}$  is a partitioned subcategory.



*Proof.* The proof is similar for all cases. Use Construction A.20 to build a candidate RPO  $(\vec{B}, B)$ . The interfaces defined by the construction are the same as for the construction of RPOs in the full s-category  $\text{'BIG}(\Sigma_K)$  [118]. Therefore, given any other relative bound  $(\vec{C}, C)$  in the subcategory, there is exactly one arrow  $F$  of  $\text{'BIG}(\Sigma_K)$  from the codomain of  $\vec{B}$  to the codomain of  $\vec{C}$  such that  $F \circ B_i = C_i, i \in \{0, 1\}$ . We need to prove that  $F$  is an arrow of the subcategory.

1. By Lemma A.15.
2. By Lemma A.15.

3. This is a full subcategory.
4. This follows as  $F$  is well-sorted.

□

**Proposition A.22.** *Bounded complete and controlled full subcategories of kind sortings create RPOs along their pure functor.*

*Proof.* Let the functor  $\mathcal{I}$  define a bounded complete full subcategory  $\mathbf{A}$  of a kind s-category  $\mathbf{BIG}(\Sigma_K)$  and let  $(D_0, D_1)$  be a bound for  $(A_0, A_1)$  in the subcategory. Let the  $\mathcal{UI}$ -image of the bound have a pure RPO  $(B'_0, B'_1, B')$ . Let  $Int$  and  $Ctrl$  respectively denote the sets of interface sorts and control sorts of  $\mathbf{A}$ . We will first construct a relative bound for  $(A_0, A_1)$  relative to  $(D_0, D_1)$  in  $\mathbf{A}$  and then show that it is universal.

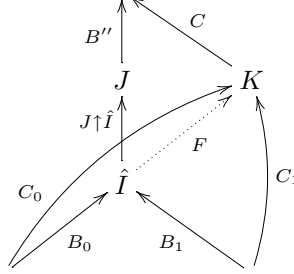
As kind sorted s-categories create RPOs, there is a corresponding sorted RPO  $(B_0, B_1, B)$  with interface  $\hat{I}$  in  $\mathbf{BIG}(\Sigma_K)$ . By the RPO construction for kind sorted bounds [118], the sort of each root  $\hat{r}$  in  $\hat{I}$  is defined by:

$$kind(\hat{r}) = \bigcup_{r <_{B_i} \hat{r}, i \in \{0,1\}} kind(r_0).$$

From the RPO construction, we have  $B(\hat{r}) = p$ , for some node or root  $p$  of  $D_0$  and  $D_1$ . As  $D_0$  and  $D_1$  are sorted and  $r <_{D_i} p$  for all  $r <_{B_i} \hat{r}$ ,  $kind(p)$  is an upper bound for  $S = \{r \mid r <_{B_i} \hat{r}, i \in \{0,1\}\}$  in  $(Int \cup Ctrl, \subseteq)$ . As  $\mathbf{A}$  is controlled, the visible controls of  $kind(p)$  form an interface sort  $\theta \subseteq kind(p)$ . As  $kind(\hat{r})$  contains no invisible controls,  $\theta$  is a bound for  $S$  in  $(Int, \subseteq)$ . As  $\mathbf{A}$  is bounded complete, there exists a join  $\theta_{\hat{r}}$  for the set  $S$  in  $(Int, \subseteq)$ . We have  $kind(\hat{r}) \subseteq \theta_{\hat{r}} \subseteq \theta \subseteq kind(p)$ .

We define the inflation  $J \upharpoonright \hat{I}$  pointwise on places  $\hat{r}$  of  $\hat{I}$  as the join  $\theta_{\hat{r}}$  of  $\{r \mid r <_{B_i} \hat{r}, i \in \{0,1\}\}$  in  $(Int, \subseteq)$ . From the above, if  $\hat{r} <_B p$  then  $kind(\hat{r}) \subseteq \theta_{\hat{r}} \subseteq kind(p)$ .  $J$  is an object of  $\mathbf{A}$  as the sorts of places of  $J$  are elements of  $(Int, \subseteq)$ . Therefore, we can factor  $B$  as  $B = B'' \circ J \upharpoonright \hat{I}$  where  $B''$  is an arrow of  $\mathbf{A}$  as its domain and codomain are objects of  $\mathbf{A}$  which is a full subcategory of the kind sorting.

We take the relative bound  $(J \upharpoonright \hat{I} \circ B_0, J \upharpoonright \hat{I} \circ B_1, B'')$  as our RPO candidate. Let  $(C_0, C_1, C)$  with interface  $K$  be a relative bound for  $(A_0, A_1)$  relative to  $(D_0, D_1)$  in  $\mathbf{A}$ . We must prove that there is a unique mediating arrow from our candidate to this relative bound. As  $(B_0, B_1, B)$  is an RPO in  $\mathbf{BIG}(\Sigma_K)$ , there is a unique mediator  $F : \hat{I} \rightarrow K$  between the RPO and  $(C_0, C_1, C)$  making the diagram below commute.



By the same argument as above, if  $\hat{r} <_F p$  then  $\text{kind}(\hat{r}) \subseteq \theta_{\hat{r}} \subseteq \text{kind}(p)$ . Therefore, we can factor  $F$  as  $F = G \circ J \uparrow \hat{I}$  where  $G$  is an arrow of  $\mathbf{A}$  by the fullness of  $\mathcal{I}$  and where  $G$  is a mediator between our candidate and  $(C_0, C_1, C)$ .  $\mathcal{U}(F)$  is a unique mediator between  $(B_0, B_1, B)$  and  $(C_0, C_1, C)$ . We have

$$\mathcal{U}(F) = \mathcal{U}(G \circ J \uparrow \hat{I}) = \mathcal{U}(G) \circ \mathcal{U}(J \uparrow \hat{I}) = \mathcal{U}(G) \circ \text{id} = \mathcal{U}(G) = \mathcal{UI}(G).$$

As  $\mathcal{UI}$  is faithful,  $G$  is therefore a unique mediator.  $\square$

## A.5 Proofs for Section 5.4

**Proposition A.23** (jointly opcartesian bounds reflect pushouts along sortings). *If  $\vec{B}$  is a jointly opcartesian bound for  $\vec{A}$  along a sorting  $\mathcal{U}$  and  $\mathcal{U}(\vec{B})$  is a pushout for  $\mathcal{U}(\vec{A})$ , then  $\vec{B}$  is a pushout for  $\vec{A}$ .*

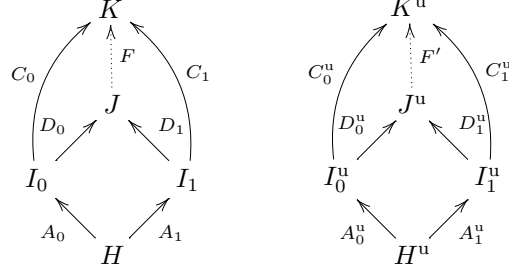
*Proof.* Let  $\mathbf{BIG}(\Sigma)$  be the domain of  $\mathcal{U}$  and let  $I$  be the outer interface of  $\vec{B}$ . Let  $\vec{C}$  be some bound for  $\vec{A}$  in  $\mathbf{BIG}(\Sigma)$  with outer interface  $J$ . We must prove that there is a unique mediator  $j : I \rightarrow J$  such that  $j \circ B_i = C_i, i \in \{0, 1\}$ .

As  $\mathcal{U}(\vec{B})$  is a pushout for  $\mathcal{U}(\vec{A})$ , there is a unique mediator  $j' : \mathcal{U}(I) \rightarrow \mathcal{U}(J)$ . As  $\vec{B}$  is a jointly opcartesian bound, there is a unique  $j$  in  $\mathcal{U}$  such that  $\mathcal{U}(j) = j'$  and  $C_i = j \circ B_i$ . Assume the existence of a second mediator  $k$  between from  $\vec{B}$  to  $\vec{C}$  in  $\mathcal{U}$ . As underlying bigraphs share the same parent and link graphs,  $\mathcal{U}$  is faithful, and  $j'$  is a unique mediator of pure bigraphs, it follows that  $k = j$ . Therefore,  $\vec{B}$  is a pushout for  $\vec{A}$ .  $\square$

**Proposition A.24.** *Whenever  $\vec{D}$  bounds  $\vec{A}$  in a subcategory  $\mathcal{I}$  of a kind sorting and  $\mathcal{UI}(\vec{D})$  is a pushout for  $\mathcal{UI}(\vec{A})$ , then  $\vec{D}$  is a pushout for  $\vec{A}$  if:*

1.  $\mathcal{I}$  is a fitting meet subcategory;

2.  $\mathcal{I}$  is a partitioned subcategory.



*Proof.* To prove that  $\vec{D}$  is a pushout for  $\vec{A}$ , we first show that for any other bound  $\vec{C}$  for  $\vec{A}$ , there exists a sorted mediator  $F$  such that  $F \circ D_i = C_i$  and then prove the uniqueness of  $F$ . The situation is depicted in the diagram above.

Assume that  $\vec{C}$  is another bound for  $\vec{A}$ . Let  $F'$  be the unique (pure) mediating arrow from  $(D_0^u, D_1^u)$  to  $(C_0^u, C_1^u)$ . Let  $F$  have the same *prnt* map as  $F'$ .

First, we must show that  $F$  is an arrow of  $\mathbf{BIG}(\Sigma_K)$  *i.e.* that it is sorted.

1. The proof follows from Lemma A.15.
2. We break the proof over the kind sorting conditions. In the following,  $v$  and  $v'$  are nodes,  $r$  is a site of  $F$ , and  $p$  is a root or node of  $F$ .

**K1** Let  $p = F(v)$ . Then  $p = C_0(v)$  and since  $C_0$  is sorted,  $F$  obeys the sorting.

**K2** Let  $p = F(r)$ . We must prove  $\text{kind}(r) \subseteq \text{kind}(p)$ .

To prove this, we will examine the construction of pushouts in pure place graphs [113].

The root  $r$  must be a parent of a site  $s \in I_0$  or a node  $v$  in  $D_0$ .

Let  $s <_{D_0} r$ . By Lemma 5.27,  $\text{kind}(r) = \text{kind}(s)$ . As  $C_0^u = F' \circ D_0^u$ ,  $s <_{C_0} p$ . Therefore,  $\text{kind}(r) = \text{kind}(s) \subseteq \text{kind}(p)$ .

Let  $v <_{D_0} r$ . Then, by the construction of pushouts,  $v <_{A_1} s' \in I_1$  and  $s' <_{D_1} r$ . By Lemma 5.27,  $\text{kind}(r) = \text{kind}(s')$ . As  $C_1^u = F' \circ D_1^u$ ,  $s' <_{C_1} p$ . Therefore,  $\text{kind}(r) = \text{kind}(s') \subseteq \text{kind}(p)$ .

**K3** Follows from  $C_0^u = F^u \circ D_0^u$ .

$F$  is a sorted mediator between the two bounds  $\vec{D}$  and  $\vec{C}$ . Lemma A.15 and the definition of partitioned s-categories imply that  $F$  is an arrow in the s-category. The uniqueness of  $F$  follows from the fact that  $\mathcal{UI}$  is faithful and that  $F'$  is a unique mediator.  $\square$

**Corollary A.25.** *If  $\vec{B}$  is a jointly opcartesian bound for  $\vec{A}$  along  $\mathcal{UI}$  where  $\mathcal{I}$  is a controlled meet full subcategory and  $\mathcal{UI}(\vec{B})$  is a pushout for  $\mathcal{UI}(\vec{A})$ , then  $\vec{B}$  is a pushout for  $\vec{A}$ .*

*Proof.* Both  $\mathcal{U}$  and  $\mathcal{I}$  are faithful. Therefore,  $\mathcal{UI}$  is faithful.  $\mathcal{UI}$  is also surjective on objects and hence is a sorting functor. The proof follows by Proposition A.23.  $\square$

## A.6 Proofs for Chapter 6

### A.6.1 Tiled link sorting

In the following, we let  $p_0$  range over  $P_0$ ,  $p_i$  over  $P_i (i = 0, 1)$ , and  $x_i$  over  $X_i, (i = 0, 1)$ . We will let  $q$  range over points of a link graph, using subscripting when it is useful. When discussing two link graphs  $\vec{A}$  with a common domain  $W$ , we let  $p_2$  range over  $P_2 \uplus W$  where  $P_2$  is the set of ports common to  $\vec{A}$ . The same conventions apply to node sets  $V$  and edge sets  $E$ .

**Proposition A.26** (composition respects tiled link sorting). *If  $A : \langle m_0, \theta_0, X_0 \rangle \rightarrow \langle m_1, \theta_1, X_1 \rangle$  and  $B : \langle m_1, \theta_1, X_1 \rangle \rightarrow \langle m_2, \theta_2, X_2 \rangle$  are sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.*

*Proof.* We first consider idle edges and names. If an edge  $e$  is idle in  $A$  or  $B$  then  $e$  is idle in  $A \circ B$  and  $\Phi$  is satisfied. If a name  $x_1$  is idle in  $A$  then it has sort **a**.  $x_1$  disappears in the composition but since it is linked in  $A$  to a link connecting only points of sort **a**, this is of no consequence. If a name  $x_2$  is idle in  $B$  then it has sort **a** and it is also idle in  $B \circ A$ .  $\Phi$  is satisfied.

We now consider the images of elements in the domain of *link*. Recall that

$$\text{link} = \text{link}_{B \circ A} = (\text{id}_{E_A} \uplus \text{link}_B) \circ (\text{link}_A \uplus \text{id}_{P_B})$$

where  $P_B$  are the ports of  $B$  and  $E_A$  are the edges of  $A$ .

We consider the case where port  $q$  has sort **d**.

- Let  $q$  be a port of  $A$  with closed link  $e$  of  $A$ . Then  $(B \circ A)(q_0^d) = e$ . Since  $A$  is sorted, there are two cases. Either  $q$  is the only point of  $e$  in  $A$  or else  $e$  also has a point of sort  $op(d)$ . The same holds in  $B \circ A$ .
- Let  $A(q) = x$ . As  $A$  is sorted,  $x$  has sort **d** and  $q$  is the only point of  $x$  in  $A$ . We have two cases:
  - $B(x) = y$ . As  $B$  is sorted,  $y$  has sort **d** and  $x$  is the only point of  $y$  in  $B$ . Therefore,  $q$  is the only point of  $y$  in  $B \circ A$ .

- $B(x) = e$ . As  $B$  is sorted, we have three cases.
  - \*  $x$  is the only point of  $e$  in  $B$  and so  $q$  is the only point of  $e$  in  $B \circ A$ .
  - \*  $e$  links one other port  $q$  with sort  $op(d)$  in  $B$ . Therefore,  $e$  links two ports of opposite direction in  $B \circ A$ .
  - \*  $e$  links one inner name  $x'$  with sort  $op(d)$  in  $B$ . As  $A$  is sorted,  $x'$  links exactly one point with direction  $op(d)$  in  $A$ . Therefore,  $e$  links two points of opposite direction in  $B \circ A$ .
- Let  $B(q) = e$ . Then  $(B \circ A)(p) = e$ . Since  $B$  is sorted, there are two cases:
  - $e$  only links  $q$  in  $B$  and hence in  $B \circ A$ .
  - $e$  links one point with sort  $op(d)$  in  $B$ . There are two cases:
    - \*  $B(p) = e$ , with  $p$  a port of  $B$ . Therefore,  $e$  links two ports of opposite direction in  $B \circ A$ .
    - \*  $B(x) = e$ . As  $A$  is sorted,  $x$  links exactly one point of sort  $op(d)$  in  $A$ . Therefore,  $e$  links two points of opposite direction in  $B \circ A$ .
- Let  $B(q) = x$ . As  $B$  is sorted,  $x$  has sort  $d$  and  $q$  is the only point of  $x$  in  $B$  and hence in  $B \circ A$ .

□

**Construction A.27** (building a tile-sorted RPO).

*Proof.* Let  $\vec{A} : W \rightarrow \vec{X}$  have a bound  $\vec{D} : \vec{X} \rightarrow Z$  in tile sorted link graphs. We wish to build a sorted RPO

$$(\vec{B} : \vec{X} \rightarrow \hat{X}, B : \hat{X} \rightarrow Z).$$

We start by building a pure RPO  $(\vec{B}', B')$  for  $\vec{A}^u$  to  $\vec{D}^u$  (Construction 2.38). From this we shall construct a sorted bound  $(\vec{B}, B)$  for  $\vec{A}$  relative to  $\vec{D}$ , such that  $(\vec{B}, B)^u = (\vec{B}', B')$ . Then in the next proposition we shall show that it is a sorted RPO.

Let  $X'$  be the interface of this pure RPO. We will construct a sorted interface  $\hat{X}$  with  $\hat{X}^u = X'$  by ascribing a sort to each  $x \in X'$  during the proof. Let  $\vec{B}$  be the sorted lift of  $\vec{B}'$  with codomain  $\hat{X}$ . The first step is to show that  $B_0$  and  $B_1$  are sorted. Then proof then follows from Lemma 6.13.

We first examine the construction of  $X'$ ,  $B'_0$ , and  $B'_1$  and show that for any point  $q_0$  in  $B_0$  such that  $B_0(q_0) = x \in \hat{X}$ , the sorting is obeyed. The proof for  $B_1$  is similar.



Recall that the construction defines two sets

$$X'_i \stackrel{\text{def}}{=} \{x \in X_i \mid D_i(x) \in E_3 \uplus Z\}$$

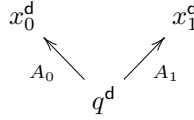
and defines the smallest equivalence relation  $\cong$  on  $X'_0 + X'_1$  such that  $(0, x_0) \cong (1, x_1)$  whenever  $A_0(p) = x_0$  and  $A_1(p) = x_1$  for some point  $p \in W \uplus P_2$ . The RPO interface  $X'$  is then defined up to isomorphism as

$$X' \stackrel{\text{def}}{=} (X'_0 + X'_1) / \cong.$$

We subscript members of  $X'_i$  with  $i$  and let  $\hat{x}_i$  denote the  $\cong$ -equivalence class of  $(i, x_i), x_i \in X'_i$ .

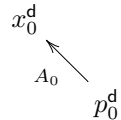
The sorting is defined as: if  $x$  maps to  $\hat{x}$  in  $B'_0$  or  $B'_1$  then  $\text{sort}(\hat{x}) = \text{sort}(x)$ . The proof that this mapping is well-defined (*i.e.* that all such  $x$  that map to  $\hat{x}$  have the same sort) and that each  $\hat{x}_i \in \hat{X}$  obeys the sorting follows. Recall that  $\text{ts}(G)$  denotes that a bigraph  $G$  obeys tiled link sorting.

- Let  $x_0^d \in X'_0$ . Then, by  $\text{ts}(A_0)$ ,  $A_0^{-1}(x_0^d) = \{q^d\}$ . We have a case split.
  - Let  $q^d \in w \uplus P_2$ .  $A_1(q^d)$  must be a name in order for  $\vec{B}^u$  to be a bound for  $\vec{A}^u$  as  $B_0(x_0^d)$  will be a name. Hence,  $A_1(q^d) = x_1^d$  and so by  $\text{ts}(A_1)$ ,



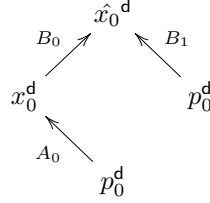
depicts  $q^d$  as the unique preimage of  $x_0^d$  and  $x_1^d$  in  $A_0$  and  $A_1$  respectively. As  $D_0(x_0^d)$  is a name or a fresh port and  $\vec{D}$  is a bound for  $\vec{A}$ ,  $x_1^d \in X'_1$ . Thus, the  $\cong$ -equivalence class  $\hat{x}_0^d$  of  $x_0^d$  and  $x_1^d$  contains exactly those two elements and is of direction  $d$ . So,  $B_i(x_i^d) = \hat{x}_0^d$  which satisfies the sorting. Also, there exists no  $p' \in P_i - P_2$  such that  $A_i(p) = x_i^d$  as we have already identified the unique preimage of those names under  $A_i$ . By the construction,  $\hat{x}_0^d$  then has exactly one preimage in each of  $B_0$  and  $B_1$ , *i.e.*  $x_0^d$  and  $x_1^d$  respectively, which is of the same direction and so  $\text{ts}(\vec{B})$ .

- Let  $q^d = p_0^d \in P_0 - P_2$  be the unique preimage of  $x_0^d$  in  $A_0$  and we can draw



By the construction,  $x_0^d$  is the only member of the  $\cong$ -equivalence class  $\hat{x}_0^d$  which is of sort  $d$  and thus is the only point of  $B_0$  that maps to  $\hat{x}_0^d$ , respecting the sorting.

$p_0^d$  is also a point in  $B_1$  and according to the construction, it is the only point in  $B_1$  that maps to  $\hat{x}_0^d$ . This information is depicted below.



Again, the sorting is obeyed and  $\text{ts}(B_0)$  for this case.

- Let  $x_0^a \in X'_0$ .

If  $A_0^{-1}(x_0^a) = \emptyset$  i.e.  $x_0^a$  is idle, then  $x_0^a$  forms its own equivalence class  $\hat{x}_0^a$  and is the only point in  $B_0$  or  $B_1$  to map to  $\hat{x}_0^a$ . Thus,  $\text{ts}(B_0)$  and  $\text{ts}(B_1)$  holds for  $\hat{x}_0^a$ .

Otherwise,  $A_0^{-1}(x_0^a) = \{q_1, \dots, q_n\}, n > 0$ . By  $\text{ts}(A_0)$ , all these points are undirected. By  $\text{ts}(A_1)$ , each shared point in  $\{q_1, \dots, q_n\}$  will map to some  $x_1^a$  in  $X_1$  and as  $D_0(x_0^a)$  is a name or a fresh port and  $\vec{D}$  is a bound for  $\vec{A}$ ,  $x_1^a \in X'_1$ . We can repeat this process of getting inverse images of undirected names and then taking the images of the undirected points until we have two subsets  $X''_0 \subset X'_0$  and  $X''_1 \subset X'_1$ , all of which contain undirected names. These elements form a  $\cong$ -equivalence class and they all map into undirected  $\hat{x}_0^a$  under  $B_0$  and  $B_1$  respectively, obeying the sorting.

Let  $p \in P_1 - P_2$ . Then if  $A_1(p) = x_1^a \in X''_1$ , by  $\text{ts}(A_1)$ ,  $p$  is undirected i.e.  $p^a$ . By the construction,  $B_0(p) = \hat{x}_0^a$ . Similarly for  $p \in P_0 - P_2$ . Hence,  $\text{ts}(B_i)$  for this case.

Thus, every outer name of  $B_0$  and  $B_1$  obeys the sorting.

Next, we examine edges in  $B_i$  to ensure that they obeys the sorting. Since  $B' \circ B'_i = D_i^u$ , by the definition of composition of link graphs,  $B_i^{-1}(e_1) = D_i^{-1}(e_1)$  for any  $e_1 \in E_1 - E_2$ . But this covers all the edges of  $B_i$ . As  $\text{ts}(D_i)$ , we immediately have that  $\text{ts}(B_i)$  for all edges in  $B_i$ . Thus,  $B_0$  and  $B_1$  are tile-sorted link graphs.

Now, we have proved that  $\vec{B}$  and  $\vec{D}$  are both sorted bounds for  $\vec{A}$  such that each  $x \in \hat{X}$  is not idle in at least one of  $(B_0, B_1)$ . There also exists a mediator  $B' = \mathcal{U}(B)$  such that  $B' \circ B'_i = D_i^u$ . Thus, by Lemma 6.13, there exists a sorted  $B' : \hat{X} \rightarrow Z$  such that  $\mathcal{U}(B') = \mathcal{U}(B)$ . As  $\mathcal{U}$  is faithful,  $B$  is this mediating arrow. Therefore,  $(\vec{B}, B)$  is a candidate RPO for  $\vec{A}$  to  $\vec{D}$ .  $\square$

### A.6.2 Plain sorting

**Proposition A.28** (valid RPO construction). *Construction 6.21 builds RPOs in  $\mathcal{SBG}_{\text{loc}}$ .*

*Proof.* Let  $(\vec{B}, B)$  be a triple created by Construction 6.21 relative to a bound  $\vec{D}$  for a span  $\vec{A}$ . We first prove that the construction yields a sorted triple. We do this by referring to the pure construction of link graph RPOs [74].

We prove that  $B_0$  is well-sorted. The proof for  $B_1$  is similar. We break the proof over the cases in the construction.

- Let  $x \in X_0$ .
  - If  $x \in X'_0$  then  $B_0(x) = \widehat{0, x}$  and  $\text{type}(x) = \text{type}(\widehat{0, x})$  by construction.
  - If  $x \notin X'_0$  then  $B_0(x) = p_1$  is a binding port where  $p_1 \in P_1 - P_2$ . Then  $D_0(x) = p_1$  and since  $D_0$  is sorted,  $\text{type}(x) = \text{type}(p_1)$ .
- Let  $p \in P_1 - P_2$ .
  - If  $B_0(p) = \widehat{1, x}$  then  $A_1(p) = x$ . As  $A_1$  is sorted, we have  $\text{type}(p) = \text{type}(x) = \text{type}(\widehat{1, x})$  from the construction.
  - If  $B_0(p) = p'$  then  $D_0(p) = p'$  and as  $D_0$  is sorted,  $\text{type}(p) = \text{type}(p')$ .

Neither  $B_0$  nor  $B_1$  have idle names. Therefore, by Lemma 6.20,  $B$  is well-sorted. Therefore  $(\vec{B}, B)$  is a relative bound.

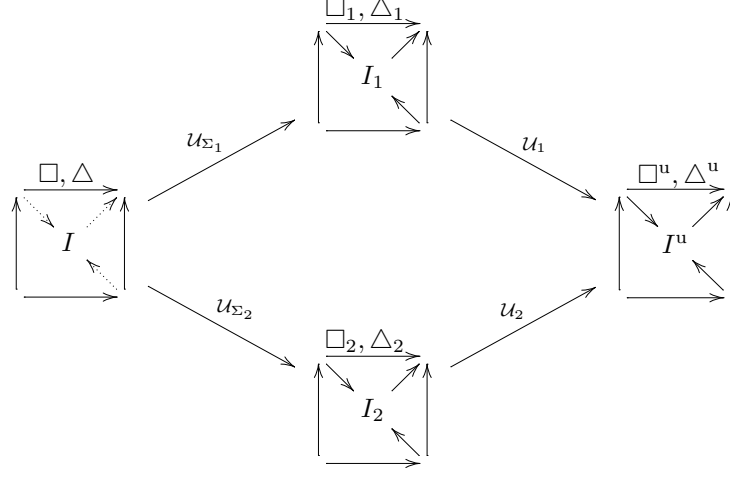
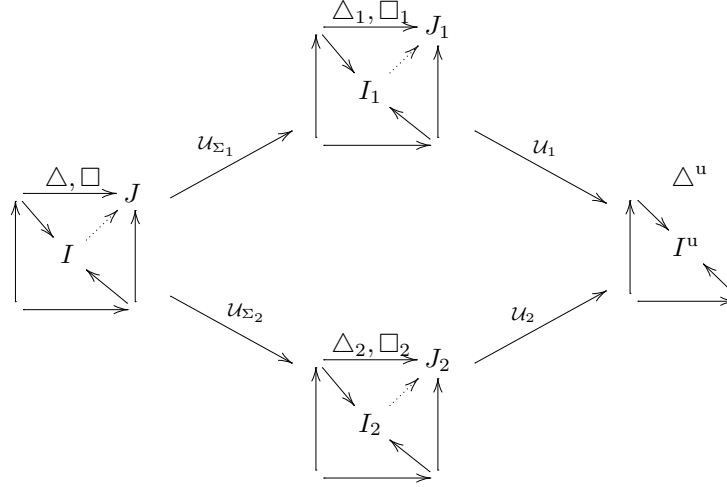
Let  $(\vec{C}, C)$  be a candidate RPO with mediating interface  $K$ . There is a unique mediator  $\mathcal{U}_{\text{type}}(K) : \mathcal{U}_{\text{type}}(I) \rightarrow \mathcal{U}_{\text{type}}(C)$ . As neither  $B_0$  nor  $B_1$  have idle names, there is then a unique sorted mediator  $K : I \rightarrow C$  by Lemma 6.20. Hence, Construction 6.21 builds an RPO.  $\square$

### A.6.3 Compound sortings

**Proposition A.29.** *If the functors  $\mathcal{U}_i : \mathcal{BIG}(\Sigma_i) \rightarrow \mathcal{BIG}(\mathcal{K})$  create RPOs then so does  $\mathcal{U}_i \circ \mathcal{U}_{\Sigma_i}$ .*

*Proof.* To save space, we will use the symbols  $\square$  and  $\triangle$  to respectively represent a commuting square and a relative bound/RPO for the square in  $\mathcal{BIG}(\Sigma_1 \times_{\mathcal{K}} \Sigma_2)$  and talk about the image of these shapes under different functors. The situation is depicted in Figure A.3.

The square  $\mathcal{U}_i(\mathcal{U}_{\Sigma_i}(\square)) = \square^u$  has a pure RPO  $\triangle^u$ . Therefore, there are sorted RPOs  $\triangle_1$  and  $\triangle_2$  such that  $\mathcal{U}_1(\triangle_1) = \mathcal{U}_2(\triangle_2) = \triangle^u$  where the respective interface sorts of the RPO interface are given by functions  $f_1, \dots, f_i$  and  $g_1, \dots, g_j$ . Define the candidate RPO  $\triangle$  for  $\square$  as the underlying RPO  $\triangle^u$  with the interface sort  $f_1, \dots, f_i, g_1, \dots, g_j$ .

**Figure A.3:** *Constructing a paired sorted RPO***Figure A.4:** *Pushouts are (weakly) reflected*

We first prove this triple is sorted. We have  $\mathcal{U}_{\Sigma_i}(\Delta) = \Delta_i$ . By Lemma 6.28,  $\Delta$  is a triple of arrows in  $\mathcal{B}\text{IG}(\Sigma_1 \times_{\mathcal{K}} \Sigma_2)$  and is a relative bound for  $\square$ .

Now, let  $\Delta'$  be any other relative bound for  $\square$ . As  $\Delta_i$  is an RPO, there are unique mediators  $F_i$  from  $\Delta_i$  to  $\mathcal{U}_{\Sigma_i}(\Delta')$  where  $\Phi_i(F_i)$  such that  $\mathcal{U}_1(F_1) = \mathcal{U}_2(F_2)$ . By Definition 6.27, there is a unique mediator  $F$  from  $\Delta$  to  $\Delta'$ .  $\square$

**Proposition A.30.** *If the functors  $\mathcal{U}_i : \mathcal{BIG}(\Sigma_i) \rightarrow \mathcal{BIG}(\mathcal{K})$  (weakly) reflect pushouts then so does  $\mathcal{U}_i \circ \mathcal{U}_{\Sigma_i}$ .*

*Proof.* We will prove that  $\mathcal{U}_i \circ \mathcal{U}_{\Sigma_i}$  strongly reflects pushouts. For a proof of weak reflection, replace ‘bound’ with ‘IPO’ in the argument below, noting that Proposition 6.29 implies that the  $\mathcal{U}_{\Sigma_i}$ -images of an IPO are IPOs.

Figure A.4 depicts the situation as in the last proposition. This time,  $\triangle$  represents a bound (the lower left triangles of arrows in the diagram) which we will prove is a pushout and  $\square$  represents some arbitrary bound (the squares of arrows in the diagram). Proving  $\triangle$  is a pushout means proving that there is a unique mediator  $F : I \rightarrow J$ .

Let the bound  $\mathcal{U}_i(\mathcal{U}_{\Sigma_i}(\triangle)) = \triangle^u$  be a pure pushout. As  $\mathcal{U}_i$  reflects pushouts, the bounds  $\triangle_1$  and  $\triangle_2$  are pushouts. There are then unique mediators  $F_i : I_i \rightarrow J_i$  between  $\triangle_i$  and  $\square_i$  where  $\Phi_i(F_i)$ . These mediators have the same underlying pure bigraph since  $\triangle^u$  is a pushout. By Definition 6.27, there is a unique mediator  $F : I \rightarrow J$  between the bounds in  $\Sigma_1 \times_{\mathcal{K}} \Sigma_2$ .  $\square$

# Appendix B

## Appendix for Part II

*Two Minutes Silence* – John Lennon & Yoko Ono

This appendix contains proofs of results used in Part II. We start in Section B.1 by providing details of some propositions used in the proof of confluence for  $\Lambda_{\text{sub}}$ .

We then recall some of our previous explorations of PSN in subcalculi of  $\Lambda_{\text{sub}}$  with varying degrees of composition of substitutions. Section B.2 presents proofs – based on Bloo and Rose’s work on  $\lambda\text{xgc}$  – that certain subsets of the rewrite relation preserve strong normalisation of  $\beta$ -reduction. In Section B.2.2, we prove PSN for a subcalculus of  $\Lambda_{\text{sub}}$  without certain interactions between substitutions. We identify the subset  $\text{SN}_{\Lambda_{\text{sub}}}$  of strongly normalising terms of  $\Lambda_{\text{sub}}$  and show it to be a strict subset of the strongly normalising terms of  $\lambda\text{xgc}$ . This identification is quite inelegant and we present a much neater one (joint work with Delia Kesner) in Chapter 9.

Previously, we continued this inductive reasoning and gave a somewhat complicated proof of PSN [122]. However, we omit that work here as we find the proofs of Chapter 9 to be much clearer.

In Section B.3 we prove that  $\lambda\text{blxr}$ , our modified version of Kesner and Lengrand’s  $\lambda\text{lrx}$  calculus, has the property of preservation of strong normalisation (PSN) of  $\beta$ -reduction. The proof is based on Lengrand’s general strategy for proving PSN through simulation in  $\lambda I$  (extended with a ‘memory construct’) and his proof of PSN for  $\lambda\text{lrx}$  [93].

Our modification of  $\lambda\text{lrx}$  makes it less elegant. In order to prove PSN for  $\lambda\text{blxr}$ , we also have to introduce some inelegance into Lengrand’s original proofs.

We then present the detailed proof of simulation of  $\Lambda_{\text{sub}}$  reduction in  $\lambda\text{blxr}$ .

## B.1 Proofs for Section 8.1

**Proposition B.1.** *In  $\Lambda_{\text{sub}}$ ,  $\rightarrow_{\text{cgc}}$  SN.*

*Proof.*  $\rightarrow_{\text{cgc}}$  SN is shown by finding a map  $h : \Lambda x \rightarrow \mathbb{N}$  such that for all  $t \rightarrow_{\text{cgc}} u$  we have  $h(t) > h(u)$ . We call this map a weighting. Before we define the weighting, we introduce a labelling of terms. This labelling and proof of SN is adapted from Barendregt's book [8, Lemmas 11.12.17, 11.12.18].

For a term  $t$  in  $\Lambda x$ , we number the occurrences of variables (*i.e.* not the binders  $x$  in  $\lambda x$  or  $[x/u]$ ) in  $t$  from the right to the left, depth-first, according to the abstract syntax tree, starting with the number 0. Give the  $n^{\text{th}}$  occurrence the index  $2^n$ . *e.g.*

$$xy((\lambda z.x)[x/w \ v]) \text{ becomes } x^{16}y^8((\lambda z.x^4)[x/w^2v^1]).$$

We define the weighting as:  $h(x^n) = n, h(t \ u) = h(t) + h(u), h(\lambda x.t) = h(t), h(t[x/u]) = h(t) + h(u)$  *i.e.*  $h(t)$  is the sum of all indices in  $t$ .

Next we state two properties on labelled terms

$$\begin{aligned} Prp(t) &\stackrel{\text{def}}{=} \forall x^i \subseteq t, \text{ if } x^i \text{ is bound by } [x/v] \text{ then } i > h(v), \\ PrpH(t) &\stackrel{\text{def}}{=} h(v) > 0 \text{ for all subterms } v \text{ of } t. \end{aligned}$$

We have the following properties.

1. If  $Prp(t)$  and  $PrpH(t)$  then if  $t \rightarrow_{\text{cgc}} t'$ ,  $h(t) > h(t')$
2.  $Prp(t)$  is preserved through  $\rightarrow_{\text{cgc}}$  reduction:

- In the  $\rightarrow_{\text{gc}}$  case, the discarded substitution binds no variables.
- For the  $\rightarrow_c$  case, consider

$$t \equiv C_1[C_2[x][x/v]] \rightarrow_c C_1[C_2[v][x/v]] \equiv t'.$$

We only need consider the variables in the new copy of  $v$  – the proof follows by  $Prp(t)$ . These variables are either free in  $t'$  or else bound by some abstraction or substitution above  $C_2[v][x/v]$  as variable capture does not occur. As  $Prp(t)$ , these variables satisfy the necessary condition and so  $Prp(t')$ .

3. Proving  $PrpH(t)$  amounts to proving that all variables have a positive, non-zero label. Thus,  $PrpH(t)$  is preserved through  $\rightarrow_{\text{cgc}}$  reduction.
4. Label any term  $t$  with the initial labelling described above. We have  $Prp(t)$  and  $PrpH(t)$  and so if  $t \rightarrow_{\text{cgc}} t'$ ,  $h(t) > h(t')$ .

As  $h(t)$  is finite for all terms  $t$ , the proof then follows. □

**Corollary B.2** (substitution lemma [132]).  $t[x/u][y/v] \longleftrightarrow_{\text{cgc}} t[y/v][x/u[y/v]]$ .

*Proof.* Follows from Lemma 8.3, the  $\lambda$ -calculus substitution lemma

$$t\{x/u\}\{y/v\} \equiv t\{y/v\}\{x/u\{y/v\}\} \quad (*),$$

and as  $\downarrow_{\text{cgc}}(t)$  is a pure term for any  $t$ . Explicitly;

$$\begin{aligned} & \downarrow_{\text{cgc}}(t[x/u][y/v]) \\ \stackrel{8.3}{\equiv} & \downarrow_{\text{cgc}}(t)\{x/\downarrow_{\text{cgc}}(u)\}\{y/\downarrow_{\text{cgc}}(v)\} \\ \stackrel{(*)}{\equiv} & \downarrow_{\text{cgc}}(t)\{y/\downarrow_{\text{cgc}}(v)\}\{x/\downarrow_{\text{cgc}}(u)\{y/\downarrow_{\text{cgc}}(v)\}\} \\ \stackrel{8.3}{\equiv} & \downarrow_{\text{cgc}}(t)\{y/\downarrow_{\text{cgc}}(v)\}\{x/\downarrow_{\text{cgc}}(u[y/v])\} \\ \stackrel{8.3}{\equiv} & \downarrow_{\text{cgc}}(t[y/v][x/u[y/v]]) \end{aligned}$$

and the corollary follows. □

**Propositions B.3.** For any  $t$ ,

1. If  $t \rightarrow_{\text{gc}} u$  then  $\text{FV}(t) \supseteq \text{FV}(u)$  and  $\downarrow_{\text{cgc}}(t) \equiv \downarrow_{\text{cgc}}(u)$ .
2. If  $t \rightarrow_c u$  then  $\text{FV}(t) = \text{FV}(u)$  and  $\downarrow_{\text{cgc}}(t) \equiv \downarrow_{\text{cgc}}(u)$ .
3. If  $t \rightarrow_b u$  then  $\text{FV}(t) = \text{FV}(u)$
4. If  $t$  is garbage-free then  $\text{fv}(t) = \text{fv}(\downarrow_{\text{cgc}}(t))$ .

*Proof.* In 1 and 2,  $\downarrow_{\text{cgc}}(t) \equiv \downarrow_{\text{cgc}}(u)$  follows by  $\rightarrow_{\text{cgc}} \text{UN}$ .

1.  $\rightarrow_{\text{gc}}$  may discard some free variables, hence  $\text{FV}(t) \supseteq \text{FV}(u)$ .
2. Let  $t \equiv C'[C[x][x/v]] \rightarrow_c C'[C[v][x/v]] \equiv u$ . The occurrence of  $x$  which is replaced was bound in  $t$  so no free variables are lost. Any  $y$  which is bound in  $v$  in  $t$  is bound within  $v$  or by some binder above  $v$  in  $t$ . The same is true of the copy of  $v$  in  $u$  and so the set of free variables does not increase and  $\text{FV}(t) = \text{FV}(u)$ .



3. By definition,  $(\lambda x.t)u$  has the same set of free variables as  $t[x/u]$ .
4. There is a reduction path

$$t \equiv t'_0 \rightarrow_{\text{cgc}} t_1 \rightarrow_{\text{gc}}^= t'_1 \rightarrow_{\text{cgc}} t_2 \rightarrow_{\text{gc}}^= t'_2 \rightarrow_{\text{cgc}} \cdots \rightarrow_{\text{gc}}^= t'_n \equiv \downarrow_{\text{cgc}}(t)$$

where

- $t_{i+1} \equiv t'_{i+1}$  if  $t'_i \rightarrow_{\text{cgc}} t_{i+1}$  is a  $\rightarrow_c$  step which does not substitute for the last free occurrence of a variable *i.e.* it does not create garbage,
- $t_{i+1} \rightarrow_{\text{gc}} t'_{i+1}$  if  $t'_i \rightarrow_{\text{cgc}} t_{i+1}$  is a  $\rightarrow_c$  step which does substitute for the last free occurrence of a variable *i.e.* it creates garbage,
- all  $t'_i$  are garbage free.

We now show that  $\text{fv}(t'_i) = \text{fv}(t'_{i+1})$ . For the first case of  $\rightarrow_c$  reduction, this follows by (2) above. For the second case, the  $\rightarrow_{\text{gc}}$  discards some free variables but those variables were copied by the  $\rightarrow_c$  reduction and avoided variable capture. The proof follows by  $\rightarrow_{\text{cgc}}$  UN.  $\square$

## B.2 PSN for subcalculi of $\Lambda_{\text{sub}}$

In this section, we prove that subcalculi of  $\Lambda_{\text{sub}}$  preserve strong normalisation (PSN) of  $\beta$ -reduction. The proofs are based on Bloo and Rose's work [20, 132, 18].

Rose [132] states that the main technical reason for the existence of explicit garbage collection in  $\lambda\text{xgc}$  is that it greatly eases the proof of PSN. We try to adapt their work to prove PSN for  $\Lambda_{\text{sub}}$  which also has explicit garbage collection. We do not take on the full calculus here using their method as we have done before and we demonstrate how the approach requires some rather strong properties to work.

We follow the inductive proof of Bloo and Rose [20] which uses the technique of garbage-free reduction. Bloo [18] gives an alternative inductive proof and Bloo and Geuvers [18, 19] use the recursive path ordering (RPO) technique.

In Section B.2.1, we follow Rose and prove PSN for a calculus  $\Lambda_{\text{sub}\downarrow\text{gc}}$  where reduction, called garbage-free reduction, can be described as 'do a  $\rightarrow_b$  or a  $\rightarrow_c$  reduction followed by total garbage collection.'

Section B.2.2 introduces two subcalculi  $\Lambda_{\text{subc}\odot}$  and  $\Lambda_{\text{subc}^b}$ . They are both weak versions of  $\Lambda_{\text{sub}}$  in that the reduction relation is a subset of  $\rightarrow_{\text{bcgc}}$ , respectively omitting all or some copying between substitutions. This copying between substitutions effectively *is* compositions of substitutions in

$\Lambda_{\text{sub}}$ . We prove PSN for  $\Lambda_{\text{subc}^\flat}$  in this section by adapting Rose’s inductive proofs for  $\lambda\text{xgc}$ . We show that as  $\Lambda_{\text{subc}^\flat}$  allows some form of composition of substitutions and  $\lambda\text{xgc}$  does not<sup>1</sup>, the set of strongly normalising terms of  $\Lambda_{\text{subc}^\flat}$  is a subset of the set of strongly normalising terms of  $\lambda\text{xgc}$ .

Section B.2.3 discusses why the property  $\text{subSN}$  used in the proof of PSN for  $\Lambda_{\text{subc}^\flat}$  is not a sufficient property for reasoning about infinite reduction paths inside garbage in  $\Lambda_{\text{sub}}$ . As might be expected, the problem is with copying between substitutions. We show that these reductions (that we disallowed in  $\Lambda_{\text{subc}^\circ}$ ) conspire to make the set of strongly normalising terms of  $\Lambda_{\text{sub}}$  a strict subset of that of both  $\lambda\text{xgc}$  and  $\Lambda_{\text{subc}^\circ}$ . The issue is that more cases of infinite reductions inside garbage can occur. We identify the set of strongly normalising terms of  $\Lambda_{\text{sub}}$  but do not give a neat characterisation as in  $\Lambda_{\text{subc}^\flat}$  or  $\lambda\text{xgc}$  in this chapter (we present this in the next chapter).

### B.2.1 PSN for $\Lambda_{\text{sub}\downarrow\text{gc}}$

**Definition B.4** (garbage-free reduction).  $\rightarrow_{\text{bc}\downarrow\text{gc}}$  is  $(\rightarrow_{\text{bc}} \cdot \multimap_{\text{gc}})$ , i.e., the composition of  $\rightarrow_{\text{bc}}$  with complete garbage collection. We denote the garbage-free reduction calculus  $\Lambda_{\text{sub}\downarrow\text{gc}}$ .

Garbage-free reduction is reduction “where all garbage is removed as soon as possible” [132] i.e. as soon as we perform a  $\rightarrow_{\text{b}}$  or  $\rightarrow_{\text{c}}$ , we immediately discard any garbage. This ensures that we do not ‘waste time’ reducing garbage.

Garbage-free reduction also has a theoretical advantage. Rose notes that for  $\lambda\text{xgc}$ , “infinite reductions consist mainly of reductions inside garbage” [132]. This is true even more so for  $\Lambda_{\text{sub}}$  as  $\rightarrow_{\text{bcgc}}$  allows copying between substitutions. This leads to yet more cases of infinite reductions than in  $\lambda\text{xgc}$ . Garbage-free reduction removes garbage on creation and avoids these infinite reduction sequences. We follow Rose by proving PSN first for  $\Lambda_{\text{sub}\downarrow\text{gc}}$  and then  $\Lambda_{\text{subc}^\flat}$ .

**Remark.** Our definition of garbage-free reduction is slightly different to that of Rose. In  $\lambda\text{xgc}$ , garbage-free reduction was defined by  $\rightarrow_{\text{bx}\downarrow\text{gc}}$ . As the normal forms of  $\rightarrow_{\text{x}}$  and  $\rightarrow_{\text{cgc}}$  coincide, this would suggest that garbage-free reduction for  $\Lambda_{\text{sub}}$  should be defined as  $\rightarrow_{\text{bcgc}} \cdot \multimap_{\text{gc}}$ . That definition should work but we may instead use the smaller relation defined above as in the following proofs, any garbage-free reduction path will begin at a garbage-free term  $t$ . Therefore, any reduction path starting with  $t \rightarrow_{\text{bc}\downarrow\text{gc}} t'$  must begin with a  $\rightarrow_{\text{b}}$  or  $\rightarrow_{\text{c}}$  reduction. As  $t'$  is guaranteed to be garbage-free, the same holds for paths beginning at  $t'$ .

Alternatively, consider the  $\rightarrow_{\text{vargc}}$  reduction. This is necessary in  $\rightarrow_{\text{x}}$  for the normal forms of  $\rightarrow_{\text{x}}$  and  $\rightarrow_{\text{cgc}}$  to coincide. However, as the garbage-free reduction paths in the proof begin at

<sup>1</sup>The extension  $\lambda\text{xc}^-$  mentioned in Chapter 7 has weak composition of substitutions and retains PSN [19, 18].

garbage-free terms,  $\rightarrow_{\text{Vargc}}$  is never applied. The remaining rules of  $\rightarrow_x$  perform a seek-and-replace role which is matched by  $\rightarrow_c$  (which does not seek but does replace).

We first prove confluence and then PSN for  $\Lambda_{\text{sub}\downarrow\text{gc}}$ .

**Lemmas B.5.**

1. For all  $\Lambda_{\text{sub}}$ -terms  $t$ ,
 
$$\begin{array}{ccc} t & \xrightarrow{\text{bcgc}} & u \\ \text{gc} \downarrow & & \text{gc} \downarrow \\ \downarrow_{\text{gc}}(t) & \xrightarrow{\text{bc}\downarrow_{\text{gc}}} & \downarrow_{\text{gc}}(u) \end{array}$$
2. For garbage-free  $t$ ,
 
$$\begin{array}{ccc} t & \xrightarrow{\text{bcgc}} & u \\ & \searrow \text{gc} & \downarrow \text{gc} \\ & \text{bc}\downarrow_{\text{gc}} & \downarrow_{\text{gc}}(u) \end{array}$$

*Proof.*

1. When  $t \rightarrow_{\text{gc}} u$ , the proof follows from  $\rightarrow_{\text{gc}}$  UN. When  $t \rightarrow_{\text{bc}} u$  and the reduction occurs inside garbage then  $\downarrow_{\text{gc}}(t) \equiv \downarrow_{\text{gc}}(u)$ . Otherwise, for  $t \rightarrow_{\text{b}} u$  we have two cases depending on whether the reduction creates garbage or not. The root cases are depicted below (using Lemma 8.2.5),

$$\begin{array}{ccc} (\lambda x.v_1)v_2 & \xrightarrow{\text{b}} & v_1[x/v_2] \\ \text{gc} \downarrow & & \text{gc} \downarrow \\ (\lambda x.\downarrow_{\text{gc}}(v_1))\downarrow_{\text{gc}}(v_2) & \xrightarrow{\text{b}} & \downarrow_{\text{gc}}(v_1)[x/\downarrow_{\text{gc}}(v_2)] \xrightarrow{\text{gc}} \downarrow_{\text{gc}}(v_1) \end{array}$$

where the dotted reduction occurs when  $x \notin \text{FV}(\downarrow_{\text{gc}}(v_1))$ . So we reach  $\downarrow_{\text{gc}}(u)$  from  $\downarrow_{\text{gc}}(t)$  with one  $\rightarrow_{\text{bc}\downarrow_{\text{gc}}}$  reduction.

A  $t \rightarrow_c u$  reduction outside garbage means that a free occurrence of some variable  $x$  is not discarded. A similar diagram sketches the proof.

$$\begin{array}{ccc} C[x][x/v] & \xrightarrow{c} & C[v][x/v] \\ \text{gc} \downarrow & & \text{gc} \downarrow \\ \downarrow_{\text{gc}}(C)[x][x/\downarrow_{\text{gc}}(v)] & \xrightarrow{c} & \downarrow_{\text{gc}}(C)[\downarrow_{\text{gc}}(v)][x/\downarrow_{\text{gc}}(v)] \xrightarrow{\text{gc}} \downarrow_{\text{gc}}(C)[\downarrow_{\text{gc}}(v)] \end{array}$$

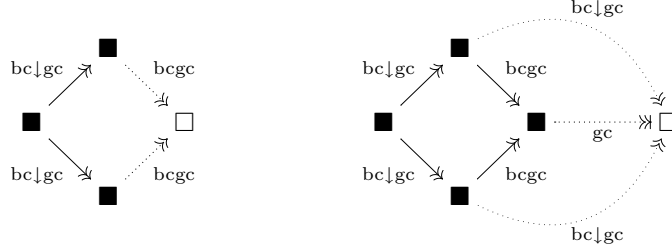
2. As  $t$  is garbage-free, the left triangle below follows from Definition B.4. The first square on the left is given by  $\rightarrow_{\text{gc}}$  UN, the next square is given by 1, and so on.

$$\begin{array}{ccccccc} t & \xrightarrow{\text{bcgc}} & t_1 & \xrightarrow{\text{gc}} & t_2 & \xrightarrow{\text{bcgc}} & t_3 & \xrightarrow{\text{gc}} & t_4 & \xrightarrow{\text{bcgc}} & \dots & \xrightarrow{\text{gc}} & u \\ & \searrow \text{bc}\downarrow_{\text{gc}} & \downarrow \text{gc} & & \downarrow \text{gc} & & \downarrow \text{gc} & & \downarrow \text{gc} & & \downarrow \text{gc} & & \downarrow \text{gc} \\ & & \downarrow_{\text{gc}}(t_1) & \xrightarrow{\text{bc}\downarrow_{\text{gc}}} & \downarrow_{\text{gc}}(t_2) & \xrightarrow{\text{bc}\downarrow_{\text{gc}}} & \downarrow_{\text{gc}}(t_3) & \xrightarrow{\text{bc}\downarrow_{\text{gc}}} & \downarrow_{\text{gc}}(t_4) & \xrightarrow{\text{bc}\downarrow_{\text{gc}}} & \dots & \xrightarrow{\text{bc}\downarrow_{\text{gc}}} & \downarrow_{\text{gc}}(u) \end{array}$$

□

**Theorem B.6.**  $\rightarrow_{bc\downarrow gc}$  CR.

*Proof.*  $\rightarrow_{bc\downarrow gc}$  CR when  $\rightarrow_{bc\downarrow gc} \diamond$  [142, Proposition 1.1.10.iv]. The latter is shown as follows. The diagram on the left below can be filled in by noting that  $\rightarrow_{bc\downarrow gc} \subseteq \rightarrow_{bcgc}$  and that  $\rightarrow_{bcgc}$  CR (which implies that  $\rightarrow_{bcgc} \diamond$ ). The diagram on the right can then be filled in by applying Lemma B.5.2 twice, noting that the terms at the starting points of the dotted reductions are garbage-free.



□

**Theorem B.7** (PSN for  $\Lambda_{\text{sub}\downarrow gc}$ ). *Pure terms that are  $\rightarrow_\beta$ -strongly normalising are also strongly normalising for  $\rightarrow_{bc\downarrow gc}$ .*

*Proof.* Assume  $t$  is pure and strongly normalising for  $\rightarrow_\beta$ . Since  $t$  is pure it has no  $\rightarrow_{cgc}$ -redexes. Thus, every  $\rightarrow_{bc\downarrow gc}$ -reduction (finite or not) starting with  $t$  is of the form  $t \equiv t_0 \rightarrow_b t_1 \rightarrow_{cgc} t_2 \rightarrow_b t_3 \rightarrow_{cgc} \dots$  where the “ $\rightarrow_{cgc}$ ” reductions are really of the form  $\rightarrow_{gc} \cdot (\rightarrow_c \cdot \rightarrow_{gc}) \dots (\rightarrow_c \cdot \rightarrow_{gc})$  as we are working in  $\rightarrow_{bc\downarrow gc}$ .

Given any such reduction, we can construct the reduction graph as below:

$$\begin{array}{ccccccc}
 t \equiv t_0 & \xrightarrow{b} & t_1 & \xrightarrow{cgc} & t_2 & \xrightarrow{b} & t_3 \xrightarrow{cgc} \dots \\
 \parallel & & \downarrow cgc & & \downarrow cgc & & \downarrow cgc \\
 \downarrow cgc(t_0) & \xrightarrow{+_\beta} & \downarrow cgc(t_1) & \xrightarrow{\dots} & \downarrow cgc(t_2) & \xrightarrow{+_\beta} & \downarrow cgc(t_3) \xrightarrow{\dots}
 \end{array}$$

where every second square starting from the leftmost square follows by Lemma 8.7.2 and where the other squares follow from  $\rightarrow_{cgc}$  UN.

$t$  is strong normalising for  $\rightarrow_\beta$  and so the lower sequence is finite. Since  $\rightarrow_{cgc}$  SN, the upper sequence must also be finite. □

### B.2.2 PSN for $\Lambda_{\text{subc}^\flat}$

**Definition** (inter-substitution reduction,  $\rightarrow_{c^\odot}$ ,  $\rightarrow_{c^\flat}$  reduction).

1. Inter-substitution reduction is the contextual closure of the reduction generated by:

$$C_1[t[y/C_2[x]]][x/u] \rightarrow_c C_1[t[y/C_2[u]]][x/u]$$

2.  $\rightarrow_{c^\odot}$  is the largest subrelation of  $\rightarrow_c$  which excludes any inter-substitution reduction.
3.  $\rightarrow_{c^b}$  is the largest subrelation of  $\rightarrow_c$  which excludes any inter-substitution reduction whose redex is not entirely located in a body of substitution.

$\rightarrow_{c^\odot}$  could also be described as excluding any  $\rightarrow_c$  reductions where the variable of the redex was located inside a substitution.  $\rightarrow_{c^b}$  could be described as excluding any  $\rightarrow_c$  reductions where the substitution definition is a top-level substitution and the variable of the redex lies inside another substitution. For example,

$$x[y/t[z/C[w]]][w/u] \rightarrow_{\text{bcgc}} x[y/t[z/C[u]]][w/u]$$

is not a  $\rightarrow_{c^\odot}$  reduction as the free occurrence of  $w$  is located inside a substitution definition. It is a  $\rightarrow_{c^b}$  reduction as the  $\rightarrow_c$  redex is entirely contained inside a body of substitution  $t[z/C[w]][w/u]^2$ . The reduction

$$t[z/C[w]][w/u] \rightarrow_{\text{bcgc}} t[z/C[u]][w/u]$$

is again not a  $\rightarrow_{c^\odot}$  reduction. It is also not a  $\rightarrow_{c^b}$  reduction as the inter-substitution copy happens between two top-level substitutions. Clearly,  $\rightarrow_{c^\odot} \subset \rightarrow_{c^b} \subset \rightarrow_c$ .

An inter-substitution reduction is a replacement of a free variable located inside a substitution definition with some term. This form of reduction is related to the notion of composition of substitutions (see Section 7.3) which has been known to break PSN in other calculi (see Sections 7.3.3, 7.3.4, and 7.3.6).

**Definition** ( $\rightarrow_{\text{bc}^\odot\text{gc}}$ ,  $\rightarrow_{\text{bc}^b\text{gc}}$ ,  $\Lambda_{\text{subc}^\odot}$ ,  $\Lambda_{\text{subc}^b}$ ). We define  $\rightarrow_{\text{bc}^\odot\text{gc}}$  as  $(\rightarrow_{\text{bgc}} \cup \rightarrow_{c^\odot})$  and  $\rightarrow_{\text{bc}^b\text{gc}}$  as  $(\rightarrow_{\text{bgc}} \cup \rightarrow_{c^b})$ . We denote their respective calculi as  $\Lambda_{\text{subc}^\odot}$  and  $\Lambda_{\text{subc}^b}$ .

$\lambda\text{xgc}$  does not have a rule to compose substitutions. Therefore, it would be reasonable to hypothesize that an inductive proof of PSN for  $\Lambda_{\text{subc}^\odot}$  would follow the inductive proof of PSN for  $\lambda\text{xgc}$ . We strongly believe this but do not prove it here. Instead, we will use the inductive proof of PSN for  $\lambda\text{xgc}$  to prove PSN for the slightly stronger  $\Lambda_{\text{subc}^b}$  calculus. However, the reasoning at each stage should also hold for  $\Lambda_{\text{subc}^\odot}$ . The reason we are able to reuse Rose's inductive proofs whilst allowing some inter-substitution reduction is that the proofs rest on a property subSN (see below) which states that bodies of substitutions are strongly normalising for  $\rightarrow_{\text{bcgc}}$ . This property must be shown to be preserved by reduction on a subset of  $\Lambda\text{x}$  which includes the strongly normalising pure terms. We will show that  $\rightarrow_{\text{bc}^b\text{gc}}$  does indeed preserve this property on a suitable subset. As

<sup>2</sup>Put another way,  $[w/u]$  is not a top-level substitution.

an intuition, note that if  $\text{subSN}(t)$  then as  $\rightarrow_{\text{bc}^\flat \text{gc}}$  does not allow any inter-substitution reduction at top-level, any other inter-substitution reduction preserves  $\text{subSN}$ .

**Lemma B.8** ( $\downarrow_{\text{cgc}} = \downarrow_{\text{c}^\flat \text{gc}} = \downarrow_{\text{c} \odot \text{gc}}$ ).

*Proof.* We prove  $\downarrow_{\text{cgc}} = \downarrow_{\text{c} \odot \text{gc}}$  which is sufficient. Given a term  $t$ , the innermost substitutions are subterms  $u[x/v]$  of  $t$  such that  $u$  and  $v$  are pure. Given such a subterm, reduce an innermost substitution:

$$t \equiv C[u[x/v]] \rightarrow_c C[u\{x/v\}[x/v]] \rightarrow_{\text{gc}} C[u\{x/v\}].$$

None of the  $\rightarrow_c$  reductions in the path are inter-substitution reductions. We may repeat this process until it ends (as  $\rightarrow_{\text{cgc}}$  SN). The proof follows by  $\rightarrow_{\text{cgc}}$  UN.  $\square$

The next two definitions help us describe if the reduction paths of a term outside garbage ( $\#gf$ ) or inside substitutions ( $\text{subSN}$ ) are finite. The intuition is that a term is strongly normalising for  $\rightarrow_{\text{bc}^\flat \text{gc}}$  if finite paths always exist both outside garbage and inside substitutions (Theorem B.13).

**Definition** ( $\#gf(t)$ ). *For all terms  $t \in \Lambda x$ , define  $\#gf(t)$  to be the maximum length of garbage-free ( $\rightarrow_{\text{bc} \downarrow \text{gc}}$ ) reduction paths starting in  $\downarrow_{\text{gc}}(t)$ .*

**Definition** ( $\text{subSN}_{\text{c} \odot}(t), \text{subSN}_{\text{c}^\flat}(t), \text{subSN}(t)$ ). *The predicates  $\text{subSN}_{\text{c} \odot}(t)$ ,  $\text{subSN}_{\text{c}^\flat}(t)$ , and  $\text{subSN}(t)$  state that all bodies of substitutions in  $t$  are strongly normalising for  $\rightarrow_{\text{bc} \odot \text{gc}}$ ,  $\rightarrow_{\text{bc}^\flat \text{gc}}$ , and  $\rightarrow_{\text{bcgc}}$  respectively.<sup>3</sup>*

We will not discuss  $\text{subSN}_{\text{c} \odot}(t)$  or  $\text{subSN}_{\text{c}^\flat}(t)$  much here. We hypothesize that the former is sufficient to prove PSN for  $\Lambda_{\text{subc} \odot}$  but the latter is not sufficient to prove PSN for  $\Lambda_{\text{subc}^\flat}$  – the term  $p[u/Z]$  in Proposition B.9 would be a counterexample to the main theorem of this section. For  $\Lambda_{\text{subc}^\flat}$ , we need the stronger property  $\text{subSN}$ .

To demonstrate how  $\#gf$  and  $\text{subSN}$  describe the finiteness of reduction sequences outside and inside garbage respectively, let  $\Omega \equiv (\lambda z.z \ z)(\lambda w.w \ w)$ ;  $\#gf(\Omega) = \infty$  and  $\text{subSN}(\Omega)$  is true (there are no substitutions) whereas  $\#gf(x[y/\Omega]) = 0$  and  $\text{subSN}(x[y/\Omega])$  is false, where  $x \neq y$ .

**Notation** (properties of terms and subsets of  $\Lambda x$ ). *We will use the same notation for a property of a term and a subset of  $\Lambda x$  e.g.  $\text{subSN}(t)$  means that  $t$  satisfies  $\text{subSN}$  whereas  $\#gf < \infty$  denotes the subset of  $\Lambda x$  which satisfies this property.*

**Definition** ( $\Lambda x^{< \infty}$ ). *We define the subset  $\Lambda x^{< \infty} = \{t \in \Lambda x \mid \forall u \subseteq t. \downarrow_{\text{cgc}}(u) \in \text{SN}_\beta\}$ <sup>4</sup>.*

<sup>3</sup>In Bloo's terminology [18], we would say that  $t$  is *decent*.

<sup>4</sup>Bloo denotes this set as  $\Lambda x^{< \infty}$  – we use  $\Lambda x^{< \infty}$  to keep our notation consistent.

The pure terms of  $\Lambda x^{<\infty}$  are exactly the strongly normalising pure terms of the  $\lambda$ -calculus. As it excludes some non-terminating  $\Lambda x$  terms, it seems a good starting point for the proof of PSN. In fact,  $\Lambda x^{<\infty}$  characterises  $\text{SN}_{\lambda\text{xgc}}$  [20] and we hypothesize that it also characterizes  $\text{SN}_{\Lambda_{\text{subc}\odot}}$ . It does not characterize  $\text{SN}_{\Lambda_{\text{subc}^b}}$  or  $\text{SN}_{\Lambda_{\text{sub}}}$  as we now show.

**Definition (Z).** We define the term  $Z \equiv z'[x/yy][y/\lambda z.zz]$  where all variables are distinct.

**Proposition B.9.**  $\text{SN}_{\Lambda_{\text{subc}^b}} \neq \text{SN}_{\lambda\text{xgc}}$ .

*Proof.* We have  $x'[w/Z] \notin \text{SN}_{\Lambda_{\text{subc}^b}}$  but  $x'[w/Z] \in \Lambda x^{<\infty}$  so  $Z \in \text{SN}_{\lambda\text{xgc}}$  [20].  $\square$

**Corollary B.10.**  $\text{SN}_{\Lambda_{\text{subc}^b}} \subset \text{SN}_{\lambda\text{xgc}}$ .

*Proof.* By definition, a  $\Lambda x$  term which is not in  $\Lambda x^{<\infty}$  has a subterm whose  $\rightarrow_{\text{cgc}}$  normal form is not strongly normalising for  $\beta$ -reduction. Any infinite  $\beta$ -reduction sequence can be matched by an infinite  $\rightarrow_{\text{bc}^b\text{gc}}$  sequence (Proposition 8.6.2 and Lemma B.8). As  $\text{SN}_{\lambda\text{xgc}} = \Lambda x^{<\infty}$ , the set  $\text{SN}_{\Lambda_{\text{subc}^b}}$  must be a subset of  $\text{SN}_{\lambda\text{xgc}}$ . Proposition B.9 proves that it is a strict subset.  $\square$

Garbage-reduction classifies all the ‘useless’ reductions which occur in garbage.

**Definition (Garbage-reduction).**

1. Garbage-reduction is the contextual closure of the reduction generated by:

- If  $u \rightarrow_{\text{bcgc}} u'$  and  $x \notin \text{FV}(\downarrow_{\text{gc}}(t))$  then  $t[x/u] \rightarrow_{\text{bcgc}} t[x/u']$  is garbage reduction.
- If  $x \notin \text{FV}(t)$  then  $t[x/u] \rightarrow_{\text{bcgc}} t$  is garbage-reduction.
- If  $x \notin \text{FV}(\downarrow_{\text{gc}}(t))$  then  $C_1[t[x/C_2[y]]][y/u] \rightarrow_{\text{bcgc}} C_1[t[x/C_2[u]]][y/u]$  is garbage reduction.

2. Reduction outside garbage is any reduction that is not garbage-reduction.

The first type of garbage-reduction reduces the term that will replace  $x$  but any free instance of  $x$  in  $t$  is contained inside garbage. The second type of garbage-reduction is simply garbage collection via  $\rightarrow_{\text{gc}}$ . The third type is special to  $\Lambda_{\text{sub}}$ . It describes the wide substitution of a variable within garbage. The context  $C_1$  in the definition is needed as the outer substitution may not be directly above the inner substitution. This garbage reduction is always a  $\rightarrow_c$  reduction and can lead to the infinite sequences in terms like  $Z$ . In  $\Lambda_{\text{subc}\odot}$ , this final type of garbage reduction does not occur. In  $\Lambda_{\text{subc}^b}$ , these reductions are a special case of the first type as the redex must be located entirely in a body of substitution.

As any  $\rightarrow_{\text{gc}}$  reduction is garbage-reduction, reduction outside garbage only pertains to the  $\rightarrow_{\text{b}}$  and  $\rightarrow_{\text{c}}$  reductions *e.g.* if  $u \rightarrow_{\text{b}} u'$  and  $x \in \text{FV}(\downarrow_{\text{gc}}(t))$  then  $t[x/u] \rightarrow_{\text{b}} t[x/u']$  is outside garbage. Any  $\rightarrow_{\text{bc}}$  reduction whose redex is not totally or partially contained in a body of substitution is also outside garbage.

**Propositions B.11.**

1. If  $t \rightarrow_{\text{bcgc}} t'$  is garbage-reduction then  $\downarrow_{\text{gc}}(t) \equiv \downarrow_{\text{gc}}(t')$ .
2. If  $t \rightarrow_{\text{bcgc}} t'$  is outside garbage then  $\downarrow_{\text{gc}}(t) \rightarrow_{\text{bc}\downarrow_{\text{gc}}} \downarrow_{\text{gc}}(t')$ . Furthermore,
  - (a) if  $t \rightarrow_{\text{c}\odot} t'$  then  $\downarrow_{\text{gc}}(t) \rightarrow_{\text{c}\odot} \Downarrow_{\text{gc}} \downarrow_{\text{gc}}(t')$ ,
  - (b) if  $t \rightarrow_{\text{c}^{\flat}} t'$  then  $\downarrow_{\text{gc}}(t) \rightarrow_{\text{c}^{\flat}} \Downarrow_{\text{gc}} \downarrow_{\text{gc}}(t')$ .

*Proof.*

1. The proof is split over the three cases where garbage-reduction occurs [122].
2. A reduction  $t \rightarrow_{\text{bc}} u$  outside garbage can be described as contracting a redex in  $t$  which exists in some garbage-free form in  $\downarrow_{\text{gc}}(t)$ . More precisely, the redex of a reduction outside garbage in a term  $t$  has a unique residual in the term  $\downarrow_{\text{gc}}(t)$ ; there is at most one residual as no copying takes place and there is at least one residual by the definition of reduction outside garbage *i.e.* the residual cannot be discarded.

The proof follows from the two diagrams in the proof of Lemma B.5.1.

- (a) If  $t \rightarrow_{\text{c}\odot} u$  then the free occurrence of the variable (say  $x$ ) in the redex does not lie under substitution. In  $\downarrow_{\text{gc}}(t)$ , it still does not lie under substitution.
- (b) If  $t \rightarrow_{\text{c}^{\flat}} u$  then the entire redex is a subterm of a body of substitution. This remains true for the reduct of the redex in  $\downarrow_{\text{gc}}(t)$ .

□

The following proofs relate specifically to  $\Lambda_{\text{subc}^{\flat}}$ . We hypothesize that they also hold for  $\Lambda_{\text{subc}\odot}$ , replacing subSN with subSN $_{\text{c}\odot}$ .

**Lemmas B.12.**

In  $\Lambda_{\text{subc}^{\flat}}$ ,

1. If subSN( $t$ ) and  $t \rightarrow_{\text{bc}^{\flat}\text{gc}} t'$  is garbage-reduction, then subSN( $t'$ ).



2. If  $\text{subSN}(t)$  then  $t$  is strongly normalising for garbage-reduction.

*Proof.*

1. This follows from the definition of garbage-reduction. For the first case, say  $t \equiv u[x/v] \rightarrow_{\text{bcgc}} u[x/v'] \equiv t'$ . Since  $\text{subSN}(t)$ ,  $v$  is strongly normalising for  $\rightarrow_{\text{bcgc}}$ -reduction. Then  $v'$  is strongly normalising for  $\rightarrow_{\text{bcgc}}$ -reduction.

For the second case, say  $t \equiv u[x/v] \rightarrow_{\text{bcgc}} u \equiv t'$ . Then the bodies of substitution of  $t'$  are a subset of those of  $t$  and so  $\text{subSN}(t')$ .

In  $\Lambda_{\text{subc}^\flat}$ , the third case is a special case of the first<sup>5</sup>.

The proof follows by induction over the structure of terms.

2. We begin by defining two interpretations for  $\text{subSN}$ -terms  $t$ . Let  $h_1(t)$  be the maximum length of  $\rightarrow_{\text{bcgc}}$ -reduction paths inside bodies of substitutions of  $t$ . The value  $h_1(t)$  is well defined as  $t$  has a finite number of substitutions and the body of each substitution is strongly normalising for  $\rightarrow_{\text{bcgc}}$ -reduction by  $\text{subSN}(t)$ . Let  $h_2(t)$  be the number of substitutions of top-level substitutions in  $t$ . Any garbage-reduction reduct of  $t$  will never have a greater number of top-level substitutions than  $t$  as a garbage-reduction of the form  $\rightarrow_{\text{b}}$  (which introduces new substitutions) will only occur under a substitution.

Let  $t \rightarrow_{\text{bcgc}} t'$  be garbage reduction occurring under a body of substitution of  $t$ . Then  $h_1(t) > h_1(t')$  as if  $h_1(t) \leq h_1(t')$  then there exists a  $\rightarrow_{\text{bcgc}}$ -reduction path starting from  $t$  which is longer than  $h_1(t)$ , the maximum such path.

Let  $t \rightarrow_{\text{bcgc}} t'$  be garbage reduction which does not occur under a body of substitution of  $t$ . By the definition of garbage reduction it must be that  $t \rightarrow_{\text{gc}} t'$  and the reduction throws away a top-level substitution. Hence, the number of top-level substitutions of  $t$  is reduced by 1 and  $h_2(t) > h_2(t')$ .

For any garbage reduction  $t \rightarrow_{\text{bcgc}} t'$  either of  $h_1(t)$  or  $h_2(t)$  decreases while the other does not increase. Hence, garbage reduction is strongly normalising for  $\text{subSN}$ -terms.

□

**Theorem B.13.** *If  $\#gf(t) < \infty$  and  $\text{subSN}(t)$  then  $t$  is strongly normalising for  $\rightarrow_{\text{bc}^\flat\text{gc}}$ -reduction.*

*Proof.* We induct over  $\#gf(t)$ .

---

<sup>5</sup>This is not true for  $\Lambda_{\text{sub}}$  and this case breaks the lemma for that calculus.

**Base case**  $\#gf(t) = 0$ . By Proposition B.11.2, any reduction  $t \rightarrow_{bc^b gc} u$  must be garbage-reduction. Now, for any garbage reduction  $t \rightarrow_{bc^b gc} u$  we have by Proposition B.11.1 that  $\downarrow_{gc}(u) \equiv \downarrow_{gc}(t)$ . Hence,  $\#gf(u) = 0$  and so by the same argument any reduction  $u \rightarrow_{bc^b gc} u'$  must also be garbage reduction. It follows that any reduction path starting at  $t$  contains only garbage reductions. As  $\text{subSN}(t)$ , it follows by Lemma B.12.2 that  $t$  is strongly normalising.

**Induction hypothesis** Suppose  $\#gf(t) > 0$ . We assume that if  $\#gf(t') < \#gf(t)$  and  $\text{subSN}(t')$  then  $t'$  is strongly normalising for  $\rightarrow_{bc^b gc}$ . We call this induction hypothesis i.h.1.

**Inductive case** Suppose there exists an infinite reduction path

$$t \equiv t_0 \rightarrow_{bc^b gc} t_1 \rightarrow_{bc^b gc} t_2 \rightarrow_{bc^b gc} t_3 \rightarrow_{bc^b gc} \dots$$

We have assumed that  $\text{subSN}(t)$ . By Lemma B.12.2,  $t$  is then strongly normalising for garbage reduction and so there is  $m$  such that  $t \rightarrow_{bc^b gc} t_m$  is garbage-reduction and  $t_m \rightarrow_{bc^b gc} t_{m+1}$  is reduction outside garbage. By Propositions B.11.1 and B.11.2, we have

$$\#gf(t_{m+1}) < \#gf(t_m) = \#gf(t) < \infty.$$

$\text{subSN}(t_m)$  by Lemma B.12. If we can prove by induction on the structure of  $t_m$  that also  $\text{subSN}(t_{m+1})$  then we can invoke i.h.1 to show that  $t_{m+1}$  is strongly normalising for  $\rightarrow_{bc^b gc}$ . We call the new induction hypothesis i.h.2. We treat some cases below, noting that  $t_m \rightarrow_{bc^b} t_{m+1}$  by definition of reduction outside garbage.

**Case**  $t_m \equiv (\lambda x.u)v \rightarrow_{bc^b gc} u[x/v] \equiv t_{m+1}$ . Bodies of substitutions in  $u$  and  $v$  are strongly normalising since they are also bodies of substitutions in  $(\lambda x.u)v \equiv t_m$  and  $\text{subSN}(t_m)$ . Also,  $\#gf(v) < \#gf((\lambda x.u)v)$  and so by i.h.1  $v$  is strongly normalising, thus  $\text{subSN}(u[x/v])$ .

**Case**  $t_m \equiv C[x][x/v] \rightarrow_{bc^b gc} C[v][x/v] \equiv t_{m+1}$ . We are in  $\Lambda_{\text{sub}c^b}$  so if this free occurrence of  $x$  is located inside a body of substitution, then the definition  $[x/v]$  is located inside the same body and the next case addresses this situation.

Otherwise,  $x$  is located outside of a body of substitution<sup>6</sup>.  $x$  is replaced by  $v$  whose bodies of substitutions are strongly normalising as they are also bodies of substitutions in  $t_m$ .

Otherwise  $t_m$  and  $t_{m+1}$  have identical bodies of substitution. Hence,  $\text{subSN}(C[v][x/v])$ .

---

<sup>6</sup>This case breaks this theorem for  $\Lambda_{\text{sub}}$  – consider the term  $t[y/x'] (\lambda z.zx)[x'/\lambda x.xx]$  and the obvious  $\rightarrow_c$  reduction which does not preserve  $\text{subSN}$ .

**Case**  $t_m \equiv u[x/v] \rightarrow_{\text{bc}^b \text{gc}} u[x/v'] \equiv t_{m+1}$  where  $v \rightarrow_{\text{bc}^b \text{gc}} v'$ . We know  $\text{subSN}(t_m)$  which implies  $\text{subSN}(u)$ . As  $\text{subSN}(t_m)$ ,  $v$  is strongly normalising for  $\rightarrow_{\text{bcgc}}$ . Hence,  $v'$  is strongly normalising and  $\text{subSN}(u[x/v'])$ .

**Case**  $t_m \equiv u \ v \rightarrow_{\text{bc}^b \text{gc}} u' \ v \equiv t_{m+1}$  where  $u \rightarrow_{\text{bc}^b \text{gc}} u'$ . Then  $\text{subSN}(u')$  by i.h.2. As  $\text{subSN}(u \ v)$ , we have  $\text{subSN}(v)$  and so  $\text{subSN}(u' \ v)$ .

**Case**  $t_m \equiv \lambda x. u \rightarrow_{\text{bc}^b \text{gc}} \lambda x. u' \equiv t_{m+1}$  where  $u \rightarrow_{\text{bc}^b \text{gc}} u'$ . Similar to the last case.

Thus,  $t_{m+1}$  is strongly normalising for  $\rightarrow_{\text{bc}^b \text{gc}}$ -reduction. This completes the proof as  $t \rightarrow_{\text{bc}^b \text{gc}}$   $t_{m+1}$  is a finite sequence.  $\square$

**Corollary B.14** (PSN for  $\Lambda_{\text{subc}^b}$ ).  $\rightarrow_{\text{bc}^b \text{gc}}$  PSN of  $\rightarrow_\beta$ .

*Proof.*

**Case**  $\Rightarrow$ . If  $t$  is pure then it has no substitutions and so  $\text{subSN}(t)$ . If  $t$  is strongly normalising for  $\rightarrow_\beta$ -reduction then by Theorem B.7,  $\# \text{gf}(t) < \infty$ . We now apply Theorem B.13.

**Case**  $\Leftarrow$ . By Proposition 8.6.2 and Lemma B.8, infinite  $\rightarrow_\beta$ -reductions induce infinite  $\rightarrow_{\text{bc}^b \text{gc}}$ -reductions.  $\square$

### B.2.3 The problem with proving PSN for $\Lambda_{\text{sub}}$

The proof in the previous section does not hold for  $\Lambda_{\text{sub}}$ . This is because inter-substitution reduction in general can introduce cases of infinite sequences inside substitution which do not occur in  $\lambda \text{xgc}$  or  $\Lambda_{\text{subc}^b}$ .

In this section, we hope to demonstrate why Bloo and Rose's strategy (which is quite elegant for  $\lambda \text{xgc}$ ) does not adapt well to  $\Lambda_{\text{sub}}$ . This is not surprising – their strategy does not consider composition of substitutions – but it may help highlight why we were not pleased with our proof of PSN using this method.

**Proposition B.15.**  $\text{SN}_{\Lambda_{\text{sub}}} \subset \text{SN}_{\Lambda_{\text{subc}^b}} \subset \text{SN}_{\lambda \text{xgc}}$ .

*Proof.* This follows from Corollary B.10.  $\square$

Bloo proposed that the crucial step in his proof of PSN for  $\lambda \text{xgc}$  was that it was provable that given  $t \in \Lambda x^{<\infty}$ , if  $\text{subSN}(t)$  then all  $\rightarrow_{\text{bxgc}}$ -reduces  $t'$  of  $t$  satisfied  $\text{subSN}(t')$ . In Proposition B.9,  $\text{subSN}(Z)$  is true and  $Z \in \Lambda x^{<\infty}$  but  $\text{subSN}(Z')$  is false and the inductive proof fails for  $\Lambda_{\text{sub}}$ .

It fails because  $\text{subSN}(t)$  is not a strong enough predicate for  $\Lambda_{\text{sub}}$ .  $\text{subSN}$  is meant to capture the property that all reduction sequences that occur inside substitutions are strongly normalising. However,  $\Lambda_{\text{sub}}$  allows inter-substitution reductions (FCS). The previous example demonstrates how substitution may alter another body of a substitution such that  $\text{subSN}$  no longer holds. In explicit substitution calculi without composition of substitutions, this behaviour is not possible. For  $\Lambda_{\text{sub}}$ , we need a stronger property than  $\text{subSN}$ .

The property we propose will consider a body of substitution and all substitutions above it, we first introduce some notation to make the remainder of the section more legible.

**Notation** ( $u[y_1 \dots y_n]$ , superbody of substitution). *When considering a subterm  $u$  of some term  $t$ , the term  $u[y_1/v_1][y_2/v_2] \dots [y_n/v_n]$  includes all the substitutions above  $u$  in  $t$  (where  $[y_i/v_i]$  lies below  $[y_{i+1}/v_{i+1}]$ ). This is abbreviated to  $u[y_1 \dots y_n]$ . When  $u$  is a body of substitution, we say  $u[y_1 \dots y_n]$  is a superbody of substitution.*

For example,  $u$  is a body of substitution in  $((\lambda z.(t_1[x/u][y_1/v_1]) t_2)[y_2/v_2] t_3)[y_3/v_3]$  and the corresponding superbody of substitution is  $u[y_1/v_1][y_2/v_2][y_3/v_3]$ .

**Definition** ( $\text{preSN}$ ). *The predicate  $\text{preSN}(t)$  states that all superbodies  $u$  of substitutions in  $t$  are strongly normalising for  $\rightarrow_{\text{bfgc}}$ .*

$\text{preSN}(Z)$  does not hold as  $(yy)[y/\lambda z.zz]$  is not strongly normalising. The predicate seems strong but we do need to consider all inter-substitutions reduction. For example, the term

$$t \equiv ((t_1[x/x_1x_3]t_2)[x_1/x_2]t_3)[x_2/\lambda y.yy][x_3/\lambda z.zz]$$

does not satisfy  $\text{preSN}(t)$  and has an infinite reduction sequence inside substitutions.

Next, we redefine the proposed set of strongly normalising terms of  $\Lambda_{\text{sub}}$  to account for inter-substitution reduction.

**Definition** ( $\Lambda_{\text{sub}}^{<\infty}$ ). *We define the subset  $\Lambda_{\text{sub}}^{<\infty} = \{t \in \Lambda_{\text{sub}} \mid \forall u \subseteq t. \downarrow_{\text{cgc}}(u[y_1 \dots y_n]) \in \text{SN}_\beta\}$ .*

Any term  $t \in \text{SN}_{\Lambda_{\text{sub}}}$  satisfies both  $\text{preSN}(t)$  and  $t \in \Lambda_{\text{sub}}^{<\infty}$ . The pure terms in  $\Lambda_{\text{sub}}^{<\infty}$  are exactly those which are strongly normalising for  $\rightarrow_\beta$ . It is therefore a likely candidate for  $\text{SN}_{\Lambda_{\text{sub}}}$  and the proof of PSN but unfortunately it contains terms which are not strongly normalising.

**Example B.16.** *These terms (whose variables are all distinct) are not strongly normalising but satisfy  $\#gf < \infty$ ,  $\text{preSN}$ , and inclusion in  $\Lambda_{\text{sub}}^{<\infty}$ .*

1.  $(\lambda x.x_1[x_2/(\lambda y.yy)x])(\lambda z.zz)$

*The  $\rightarrow_b$  reduct  $z[y/(\lambda v.vv)x][x/\lambda u.uu]$  of the term does not satisfy  $\text{preSN}$ .*

2.  $(\lambda x.x_1[x_2/(\lambda y.yy)x])[z'/t](\lambda z.zz)$

*This term reduces via a  $\rightarrow_{\text{gc}}$  reduction (discarding  $[w/p]$ ) to the last example.*

3.  $(\lambda z'.(\lambda x.x_1[x_2/(\lambda y.yy)x])z')(\lambda z.zz), y \neq z$

*This term reduces via a  $\rightarrow_{\text{b}} \rightarrow_{\text{c}} \rightarrow_{\text{gc}}$  sequence to the first example.*

In term 1, a  $\rightarrow_{\text{b}}$  reduction breaks preSN by creating a new substitution above existing ones which introduces an infinite sequence. In term 2, a similar reduction occurs but first a  $\rightarrow_{\text{gc}}$  reduction must unblock a  $\rightarrow_{\text{b}}$  redex. In term 3, a copy enables a  $\rightarrow_{\text{b}}$  redex to break preSN.

However, it is encouraging to note that the pure  $\mathbf{C}$ -images (Definition 9.29) of all the examples are not strongly normalising. i.e. the examples all arise from pure terms which are not strongly normalising.

In their inductive proofs of PSN for  $\lambda\text{xgc}$ , Bloo [18] shows that subSN is preserved by reduction for terms in  $\Lambda x^{<\infty}$  while Rose [133] shows that subSN is preserved by reduction for terms where  $\#gf < \infty$ . The examples above demonstrate that even with the more restrictive  $\Lambda x_{\text{sub}}^{<\infty}$ , preSN is not preserved by reduction.

As noted above, the problem is that a sequence of  $\rightarrow_{\text{b}}$  reductions may introduce new substitutions above existing ones and this can break preSN. We could further constrain preSN with the following definition.

**Definition** ( $\text{bigSN}_{\text{b}}$ ). *The predicate  $\text{bigSN}_{\text{b}}(t)$  states that for all sequences  $t \rightarrow_{\text{b}} t_i$ ,  $\text{preSN}(t_i)$ .*

However,  $\rightarrow_{\text{gc}}$  reductions may unblock  $\rightarrow_{\text{b}}$  redexes as in Example B.16.4 above. We would then require a stronger definition in order that preSN was preserved.

**Definition** ( $\text{bigSN}_{\text{bgc}}$ ). *The predicate  $\text{bigSN}_{\text{bgc}}(t)$  states that for all sequences  $t \rightarrow_{\text{bgc}} t_i$ ,  $\text{preSN}(t_i)$ .*

Clearly,  $\text{bigSN}_{\text{bgc}}$  is a necessary property for a term to be strongly normalising. However, now we have to prove that it is preserved under  $\rightarrow_{\text{bgc}}$  reduction. We started this investigation by weakening the condition to only consider reductions outside substitution.

**Definition** (Reduction inside substitution).

1. Reduction inside substitution is the contextual closure of the reduction generated by:

- If  $u \rightarrow_{\text{bgc}} u'$  then  $t[x/u] \rightarrow_{\text{bgc}} t[x/u']$  is reduction inside substitution.
- $C[t[x/C'[y]]][y/u] \rightarrow_{\text{bgc}} C[t[x/C'[u]]][y/u]$  is reduction inside substitution.

2. Reduction outside substitution is any other reduction.

In previous work, we proved the following lemmas [122].

**Lemma B.17** ( $\rightarrow_{\text{bgc}}$  SN).

**Lemmas B.18** (preservation, reflection of preSN).

1. If  $\text{preSN}(t)$  and  $t \rightarrow_{\text{bgc}} u$  is inside substitution then  $\text{preSN}(u)$ .
2. If  $\text{preSN}(t)$  and  $t \rightarrow_{\text{gc}} u$  is outside substitution then  $\text{preSN}(u)$ .
3. If  $\text{preSN}(t)$  and  $t \rightarrow_c u$  then  $\text{preSN}(u)$ .
4. If  $\text{preSN}(u)$  and  $t \rightarrow_b u$  is inside substitution then  $\text{preSN}(t)$ .

These lemmas allowed us to redefine  $\text{bigSN}_{\text{bgc}}$  as:

**Definition** ( $\text{bigSN}_{\text{bgc}}$ ). *The predicate  $\text{bigSN}_{\text{bgc}}(t)$  states that for all sequences  $t \rightarrow_{\text{bgc}} t_i$  containing only reductions outside substitution,  $\text{preSN}(t_i)$ .*

and then prove the following lemmas:

**Lemmas B.19** (preservation of bigSN).

1. If  $\text{bigSN}_{\text{bgc}}(t)$  and  $t \rightarrow_{\text{bgc}} t_1$  then  $\text{bigSN}_{\text{bgc}}(t_1)$ .
2. If  $\text{bigSN}_{\text{bgc}}(t)$  and  $t \rightarrow_c t_1$  does not create any new  $\rightarrow_b$  redexes outside substitution then  $\text{bigSN}_{\text{bgc}}(t_1)$ .

However, although this eases the proof/counterproof of  $\text{bigSN}_{\text{bgc}}(t)$  for an arbitrary term  $t$ , it is not true that either  $\#gf < \infty \cap \text{bigSN}_{\text{bgc}}$  or  $\Lambda x_{\text{sub}}^{<\infty} \cap \text{bigSN}_{\text{bgc}}$  is closed under reduction (Example B.16.3 is a counterexample for both). This unfortunately<sup>7</sup> led us to this heavy-handed definition.

**Definition** ( $\text{bigSN}_{\text{bcgc}}$ ). *The predicate  $\text{bigSN}_{\text{bcgc}}(t)$  states that for all sequences  $t \rightarrow_{\text{bcgc}} t_i$ ,  $\text{preSN}(t_i)$ .*

It may be possible to weaken the definition to reductions outside substitution again but we do not attempt this. It is rather unsatisfactory when compared to Bloo and Rose's proofs for  $\lambda xgc$ . Where they require a term to satisfy  $\text{subSN}$ , we require a much stronger property – the preservation of  $\text{preSN}$  through reduction. Another consequence of Example B.16.5 is that  $\text{SN}_{\Lambda_{\text{sub}}} \subset \Lambda x_{\text{sub}}^{<\infty}$  and this method does not give us a simple property with which to characterise  $\text{SN}_{\Lambda_{\text{sub}}}$ .

In this next chapter, we present proofs of PSN for  $\Lambda_{\text{sub}}$  and a neat characterisation of  $\text{SN}_{\Lambda_{\text{sub}}}$  using intersection types.

<sup>7</sup>We feel that  $\text{bigSN}_{\text{bgc}}$  is a preferable definition to  $\text{bigSN}_{\text{bcgc}}$  as  $\rightarrow_{\text{bgc}}$  SN.

### B.3 Proof of PSN for $\lambda\text{blxr}$

It seems reasonable that  $\lambda\text{blxr}$  has PSN as it differs from  $\lambda\text{lxr}$  only in the  $\rightarrow_{\text{bs}}$  rule which creates two substitutions; one as normal and one which is ‘garbage’ and can not propagate through the term. Intuitively, this does not seem to introduce new cases of infinite reductions as the garbage substitution is strongly normalising exactly when its copy is and it may only interact with substitutions above it when its copy can. We prove these intuitions here.

Our proof uses Lengrand’s strategy of proving PSN by simulating reduction of a calculus in a version of  $\lambda I$  with memory [82]<sup>8</sup>. The proof is unsurprisingly similar to the proof of PSN for  $\lambda\text{lxr}$  [93]. It transpires that just as we have altered  $\lambda\text{lxr}$ , we also have to introduce some inelegance into some of the relations and encodings that Lengrand defines.

For notational convenience, we denote the  $\lambda I$  calculus with memory simply as  $\lambda I$  and refer to the original  $\lambda I$  calculus [36] as Church’s  $\lambda I$  calculus.

**Definition B.20.** *The set  $\Lambda_I$  of terms of the  $\lambda I$ -calculus is defined by*

$$T ::= x \mid \lambda x.T \mid (T\ T) \mid [T, T]$$

where  $x \in \text{FV}(T)$  for all abstractions  $\lambda x.T$ .

**Notation** ( $\lambda I$ -terms).  $[U, T_1, T_2, \dots, T_n]$  or  $[U, \vec{T}_i]$  denote the term  $[\dots [[U, T_1], T_2], \dots, T_n]$  and  $\vec{T}_i$  denotes the application  $T_1 \dots T_n$ .

**Lemma B.21** (substitution lemma). *Given any terms  $T, U, V \in \Lambda_I$ , we have  $T\{x \setminus U\} \in \Lambda_I$  and  $T\{x \setminus U\}\{y \setminus V\} = T\{y \setminus V\}\{x \setminus U\{y \setminus V\}\}$  so long as there is no variable capture.*

The reduction rules of  $\lambda I$  are:

$$\begin{aligned} (\beta) \quad (\lambda x.T) U &\longrightarrow T\{x \setminus U\} \\ (\pi) \quad [T, V] U &\longrightarrow [T\ U, V] \end{aligned}$$

**Proposition B.22** (Church’s Theorem). *In  $\lambda I$ ,*

$$T \in \text{SN}_{\beta\pi} \Leftrightarrow T \in \text{WN}_{\beta\pi} \Leftrightarrow \forall S \subseteq T, S \in \text{WN}_{\beta\pi}.$$

*Proof.*  $\lambda I$  is a substructure of a definable extension of Church’s  $\lambda I$  calculus. The proof follows from Klop’s dissertation [82, Corollary I.7.5]<sup>9</sup>.  $\square$

<sup>8</sup>This has been referred to as Church-Klop’s  $\lambda I$ -calculus [93] and is denoted as  $\lambda I_{[\cdot, \cdot]}$  in Klop’s thesis [82].

<sup>9</sup>More generally,  $\lambda I$  is a regular/orthogonal, non-erasing combinatory reduction system [82]. Klop provides a further generalisation of Church’s Theorem for this case [ibid., Theorem 5.9.3].

### B.3.1 Proof strategy

The notion of PSN is defined for any calculus which extends the syntax of the  $\lambda$ -calculus.  $\lambda\text{lr}$  (and hence  $\lambda\text{blxr}$ ) is not an extension as the terms are required to be linear. PSN is defined for  $\lambda\text{lr}$  as meaning that “every strongly normalisable  $\lambda$ -term is encoded into a strongly normalisable  $\lambda\text{lr}$ -term” [79]. We adopt this definition and Lengrand’s proof strategy for our proof. The proof strategy is as follows, using relations depicted in Figure B.1. In Section B.3.2 we:

1. Define a relation  $\mathcal{J}$  between  $\lambda\text{blxr}$ -terms and  $\lambda I$ -terms.
2. Prove that  $\rightarrow_{\text{xr}}$  is weakly simulated through  $\mathcal{J}$  and  $\rightarrow_{\text{bs}}$  is strongly simulated through  $\mathcal{J}$ . As  $\rightarrow_{\text{xr}}$  SN, strong normalisation is reflected back through the relation w.r.t.  $\rightarrow_{\lambda\text{blxr}}$  and  $\rightarrow_{\beta\pi}$ .

In Section B.3.3 we:

3. Define an encoding  $\mathcal{B}$  from  $\lambda$ -terms to  $\lambda\text{blxr}$ -terms.
4. Define an encoding  $j$  from  $\lambda$ -terms to  $\lambda I$ -terms.
5. Prove that  $\mathcal{B}(t) \mathcal{J} j(t)$ .

In Section B.3.4 we:

6. Prove that  $j$  preserves strong normalisation with respect to  $\rightarrow_{\beta}$  and  $\rightarrow_{\beta\pi}$ .

We then conclude PSN: given any strongly normalising  $\lambda$ -term  $t$ ,  $j(t)$  is strongly normalising (step 6) and the  $\lambda\text{blxr}$  encoding  $\mathcal{B}(t)$  is related to  $j(t)$  (step 5). As strong normalisation is reflected back through  $\mathcal{J}$  (step 2), it follows that  $\mathcal{B}(t)$  is strongly normalising.

The general strategy is depicted quite succinctly by Lengrand [93]. Our proof has been published previously [120, 121] so we refer the reader to that work for the proofs, most of which we omit here. Here, we have decided to concentrate only on the changes we needed to make to Lengrand’s work.

### B.3.2 Simulation of $\lambda\text{blxr}$ in $\lambda I$

The proof of PSN for  $\lambda\text{lr}$  used a relation between  $\lambda\text{lr}$  terms and  $\Lambda_I$ .

**Definition B.23.** *The relation  $\mathcal{I}$  between well-formed  $\lambda\text{lr}$ -terms and  $\Lambda_I$  is given by the following rules:*

$$\frac{}{x \mathcal{I} x} \quad \frac{t \mathcal{I} T}{\lambda x.t \mathcal{I} \lambda x.T} \quad \frac{t \mathcal{I} T \quad u \mathcal{I} U}{t \ u \ \mathcal{I} \ T \ U} \quad \frac{t \mathcal{I} T}{t \mathcal{I} [T, N]} \ N \in \Lambda_I$$



$$\frac{t \mathcal{I} T \quad u \mathcal{I} U}{t[x/u] \mathcal{I} T\{x \setminus U\}} \quad \frac{t \mathcal{I} T}{C_x^{y,z}(t) \mathcal{I} T\{y \setminus x\}\{z \setminus x\}} \quad \frac{t \mathcal{I} T}{W_x(t) \mathcal{I} T} \quad x \in \text{FV}(T)$$

This relation is non-deterministic. The fourth rule allows arbitrary  $\Lambda_I$  terms to be ‘tacked onto’ a  $\Lambda_I$ -term  $T$  which is related to some  $\lambda\text{lxr}$ -term  $t$ . This non-determinism has to be accounted for in the proofs which is where the vector notation  $\vec{N}$  comes in handy. Generally, we will not care what the contents of  $\vec{N}$  actually are. They represent some arbitrary terms added in by the relation.

**Example B.24** (non-determinism and proof trees). *Let  $t \mathcal{I} T$  and  $u \mathcal{I} U$ . The  $\Lambda_I$ -terms which are related to  $(\lambda x.t)[y/u]$  take the form  $[(\lambda x.T)\{y \setminus U\}, \vec{M}]$  as seen by the proof tree below.*

$$\frac{\frac{\frac{t \mathcal{I} T}{\lambda x.t \mathcal{I} \lambda x.T}}{\lambda x.t \mathcal{I} [\lambda x.T, \vec{M}']}}{(\lambda x.t)[y/U] \mathcal{I} [\lambda x.T, \vec{M}']\{y \setminus U\}} \quad u \mathcal{I} U$$

*This last  $\lambda I$ -term is equivalent to  $[(\lambda x.T)\{y \setminus U\}, \vec{M}]$ .*

This non-determinism is frequently employed to derive the terms related to  $\lambda\text{lxr}$  terms containing weakenings. Consider the term  $W_x(t)$  where  $x \notin t$  by linearity. Let  $t \mathcal{I} T$  and  $x \notin \text{FV}(T)$ . In order to find a term related to  $W_x(t)$  by  $\mathcal{I}$ , we first have to introduce a free  $x$  into  $T$  like:

$$\frac{\frac{t \mathcal{I} T}{t \mathcal{I} [T, x]}}{W_x(t) \mathcal{I} [T, x]}$$

However, this relation  $\mathcal{I}$  is not sufficient to strongly simulate  $\rightarrow_{\text{bs}}$ .

**Proposition B.25.**  $\rightarrow_{\text{bs}}$  is not strongly simulated by  $\rightarrow_{\beta\pi}^+$  through  $\mathcal{I}$ .

*Proof.* We will give a counterexample. Let

$$t \equiv (\lambda x.p)u, \quad p \mathcal{I} P, \quad u \mathcal{I} U, \quad \text{and } t \mathcal{I} [(\lambda x.P, \vec{R})U, \vec{S}] \equiv T.$$

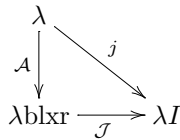


**Figure B.1:** The relations between the  $\lambda$ -calculus,  $\lambda\text{blxr}$ , and  $\lambda I$

Let  $x' \notin \text{FV}(t) \cup \text{BV}(t) \cup \text{FV}(\vec{R})$ . Now consider

$$t \equiv (\lambda x.p)u \rightarrow_{\text{bs}} C_{\Theta}^{\Gamma, \Psi}((W_{x'}(p[x/R_{\Gamma}^{\Theta}(u)]))[x'/R_{\Psi}^{\Theta}(u)]) \equiv t', \quad \Theta = \text{FV}(u).$$

We show that there is  $T' \in \Lambda_I$  such that  $t' \mathcal{I} T'$  and  $T$  cannot  $\rightarrow_{\beta\pi}^+$ -reduce to  $T'$ .

First,  $R_{\Gamma}^{\Theta}(u) \mathcal{I} R_{\Gamma}^{\Theta}(U)$  and  $R_{\Psi}^{\Theta}(u) \mathcal{I} R_{\Psi}^{\Theta}(U)$  [93]. We have  $T' = [P\{x \setminus U\}, \vec{M}, C[U], \vec{N}]$  [120]. In particular,  $T'$  contains an occurrence of  $U$  outside of  $P\{x \setminus U\}$ ,  $\vec{M}$ , or  $\vec{N}$ .

We will try to simulate the reduction  $t \rightarrow_{\text{bs}} t'$  in  $\lambda I$  by starting with  $T \equiv [[\lambda x.P, \vec{R}]U, \vec{S}] \rightarrow_{\pi}^+ [(\lambda x.P)U, \vec{R}, \vec{S}] \rightarrow_{\beta} [P\{x \setminus U\}, \vec{R}, \vec{S}]$ . It is not always true that  $U$  is not a subterm of  $\vec{R}$  or  $\vec{S}$ . However, in this case, the  $\rightarrow_{\text{bs}}$  reduction cannot be simulated as  $U$  is always present outside of  $P\{x \setminus U\}$  in  $T'$ .  $\square$

This counterexample is sufficient to disallow  $\mathcal{I}$  for our purposes. The problem is that our  $\rightarrow_{\text{bs}}$  rule creates two copies of an explicit substitution whenever it fires. The relation  $\mathcal{I}$  then always introduces  $U$  as a subterm of the  $\Lambda_I$  term corresponding to the garbage substitution bounded by the fresh  $x'$ . Our solution is simply to add some redundancy into the relation  $\mathcal{I}$ .

**Definition B.26.** *The relation  $\mathcal{J}$  between well-formed  $\lambda\text{blxr}$ -terms and  $\Lambda_I$  is given by the following rules:*

$$\begin{array}{c} \frac{}{x \mathcal{J} x} \quad \frac{t \mathcal{J} T}{\lambda x.t \mathcal{J} \lambda x.[T, x]} \quad \frac{t \mathcal{J} T \quad u \mathcal{J} U}{tu \mathcal{J} TU} \quad \frac{t \mathcal{J} T}{t \mathcal{J} [T, N]} \quad N \in \Lambda_I \\[10pt] \frac{t \mathcal{J} T \quad u \mathcal{J} U}{t[x/u] \mathcal{J} T\{x \setminus U\}} \quad \frac{t \mathcal{J} T}{C_x^{y,z}(t) \mathcal{J} T\{y \setminus x\}\{z \setminus x\}} \quad \frac{t \mathcal{J} T}{W_x(t) \mathcal{J} T} \quad x \in \text{FV}(T) \end{array}$$

The only rule we have changed is the one for abstractions. We require that an abstraction in a related  $\Lambda_I$  term carries around a free occurrence of  $x$  ‘tacked on.’ This is a redundant feature since  $x \in \text{FV}(T)$  in the rule (Lengrand shows that  $x \in \text{FV}(t)$  and  $t \mathcal{I} T$  implies  $x \in \text{FV}(T)$  [93]) but we require this redundancy, having introduced it into  $\lambda\text{blxr}$ .

The following propositions are adapted from Lengrand’s work.

**Lemma B.27.** *If  $t \mathcal{J} M$ , then*

1.  $\text{FV}(t) \subseteq \text{FV}(M)$
2.  $M \in \Lambda_I$
3.  $x \notin \text{FV}(t)$  and  $N \in \Lambda_I$  implies  $t \mathcal{J} M\{x \setminus N\}$
4.  $t \equiv t'$  implies  $t' \mathcal{J} M$
5.  $R_{\Delta}^{\Gamma}(t) \mathcal{J} R_{\Delta}^{\Gamma}(M)$

*Proof.* 1 is proved by induction on the proof tree. 2 is proved by induction, using Lemma B.21. 3 and 5 are proved by induction, noting that we do not add any new free variables with our rule. The proof of 4 follows Lengrand’s.  $\square$

**Theorem B.28** (Simulation in  $\lambda I$ ).

1. If  $t \mathcal{J} T$  and  $t \rightarrow_{\text{xr}} t'$ , then  $t' \mathcal{J} T$ .
2. If  $t \mathcal{J} T$  and  $t \rightarrow_{\text{bs}} t'$ , then there is  $T' \in \Lambda_I$  such that  $t' \mathcal{J} T'$  and  $T \rightarrow_{\beta\pi}^+ T'$ .

**Corollary B.29.** If  $t \mathcal{J} T$  and  $T \in \text{SN}_{\beta\pi}$  then  $t \in \text{SN}_{\lambda\text{blxr}}$ .

*Proof.* Proof by contradiction [93] based on termination of  $\rightarrow_{\text{xr}}$  (Lemma 7.1) and Theorem B.28.  $\square$

### B.3.3 Encoding the $\lambda$ -calculus in $\lambda I$ and $\lambda\text{blxr}$

Kesner and Lengrand give an encoding of the  $\lambda$ -calculus in  $\lambda\text{lr}$ .

**Definition B.30** ([79]). *The encoding of  $\lambda$ -terms is defined by induction as follows:*

$$\begin{aligned}
 \mathcal{B}(x) &:= x \\
 \mathcal{B}(\lambda x.t) &:= \lambda x.\mathcal{B}(t) && \text{if } x \in \text{FV}(t) \\
 \mathcal{B}(\lambda x.t) &:= \lambda x.W_x(\mathcal{B}(t)) && \text{if } x \notin \text{FV}(t) \\
 \mathcal{B}(tu) &:= C_{\Phi}^{\Delta, \Pi}(R_{\Delta}^{\Phi}(\mathcal{B}(t)) R_{\Pi}^{\Phi}(\mathcal{B}(u))) && \text{where } \Phi := \text{FV}(t) \cap \text{FV}(u)
 \end{aligned}$$

The encoding adds only the necessary details to ensure linearity – the weakening ensures that a free occurrence of  $x$  lies beneath  $\lambda x$  and the contraction renames the shared variables of  $t$  and  $u$  so that the resulting term is linear.

Lengrand provides an encoding of the  $\lambda$ -calculus into  $\Lambda_I$ .

**Definition B.31** ([93]). *We can encode the  $\lambda$ -calculus into  $\lambda I$  as follows:*

$$\begin{aligned}
 i(x) &= x \\
 i(\lambda x.t) &= \lambda x.i(t) && x \in \text{FV}(t) \\
 i(\lambda x.t) &= \lambda x.[i(t), x] && x \notin \text{FV}(t) \\
 i(tu) &= i(t)i(u)
 \end{aligned}$$

This encoding is intended for use in Lengrand’s general strategy for proving normalisation properties via simulation in  $\lambda I$ . It is the most sensible encoding, only adding anything new when required. The encoding of an abstraction  $\lambda x.t$  where  $x \notin \text{FV}(t)$  necessarily adds a free occurrence of  $x$ , required by the grammar defining  $\Lambda_I$ . However,  $i$  fails in the face of idiocy – and  $\lambda\text{blxr}$  is not a sensible calculus!

The proof strategy requires that  $\mathcal{B}(u) \mathcal{J} i(u)$  but while  $\mathcal{B}(u) \mathcal{I} i(u)$  [93], our relation  $\mathcal{J}$  breaks the proof in the case where expected – in the inductive case involving an abstraction.

**Proposition B.32.** *There exists a  $\lambda$ -term  $u$  such that  $\mathcal{B}(u)$  is not related by  $\mathcal{J}$  to  $i(u)$ .*

*Proof.* Assume  $\mathcal{B}(t) \mathcal{J} i(t)$ . Let  $u = \lambda x.t$ . Whether  $x \in \text{FV}(t)$  or not, we can not relate  $\mathcal{B}(u)$  to  $i(u)$  using  $\mathcal{J}$ , as the proof trees below suggest. The simplest example is  $u = \lambda x.x$  which can be related to  $\lambda x.[x, x]$  but not  $\lambda x.x = i(u)$ .

$$\frac{\mathcal{B}(t) \mathcal{J} i(t)}{\lambda x.\mathcal{B}(t) \mathcal{J} \lambda x.[i(t), x]} \quad \frac{\frac{\mathcal{B}(t) \mathcal{J} i(t)}{\mathcal{B}(t) \mathcal{J} [i(t), x]}}{\frac{W_x(\mathcal{B}(t)) \mathcal{J} [i(t), x]}{\lambda x.W_x(\mathcal{B}(t)) \mathcal{J} \lambda x.[i(t), x, x]}}$$

□

There are two clear ways to address this problem<sup>10</sup> – either redefine  $\mathcal{B}$  or  $i$ . Our intuition is that the latter is simpler and more viable and the proof trees in the proposition above suggest the solution – add a redundant  $x$  into both  $i$ -encodings of abstractions.

**Definition B.33** ([93]). *We encode the  $\lambda$ -calculus into  $\lambda I$  as follows:*

$$\begin{aligned} j(x) &= x \\ j(\lambda x.t) &= \lambda x.[j(t), x] & x \in \text{FV}(t) \\ j(\lambda x.t) &= \lambda x.[j(t), x, x] & x \notin \text{FV}(t) \\ j(tu) &= j(t)j(u) \end{aligned}$$

This now allows us to prove our relationship.

**Theorem B.34.** *For any  $\lambda$ -term  $u$ ,  $\mathcal{B}(u) \mathcal{J} j(u)$ .*

*Proof.* By induction on  $u$ . □

The proof of PSN for  $\lambda\text{lxr}$  utilised the fact that  $i$  preserved strong normalisation *i.e.* if  $t \in \text{SN}_\beta$  then  $i(t) \in \text{SN}_{\beta\pi}$ . As we are not using  $i$ , we need to prove the same proposition for  $j$ .

### B.3.4 The encoding $j$ preserves strong normalisation

We prove that  $j$  preserves strong normalisation by adapting Lengrand’s proofs with one difference. We omit the typing of  $\lambda$ -abstractions and  $\Pi$ -types which his proofs take into account and concentrate on the special case with no types. In the following,  $\text{nf}^\beta$  (resp.  $\text{nf}^{\beta\pi}$ ) denotes the set of  $\lambda$ - (resp.  $\lambda I$ -) terms which are in  $\beta$ - (resp.  $\beta\pi$ -) normal form.

**Lemma B.35.** *For any  $\lambda$ -terms  $t$  and  $u$ ,*

<sup>10</sup>Another solution suggested by Stéphane Lengrand seems to lead to a quicker proof of PSN by reusing previous results by Klop. It involves changing both  $\mathcal{J}$  and  $i$  and is briefly discussed in Section B.3.6.

1.  $\text{FV}(j(t)) = \text{FV}(t)$
2.  $j(t)\{x \setminus j(u)\} = j(t\{x \setminus u\})$

*Proof.* Proof by induction over the structure of  $t$ . In the second case, our alteration to  $i$  adds extra occurrences of some variable  $x$  which are bound by an abstraction in the term and are hence unaffected by substitution.  $\square$

**Definition B.36.** Let  $\sim_{\beta\pi}$  be the smallest reflexive, transitive relation on  $\Lambda_I$  containing the relation  $T \mathbf{R} U$  if  $U \rightarrow_{\beta\pi} T$ .

A term  $T$  is  $\sim_{\beta\pi}$ -related to any term which can  $\rightarrow_{\beta\pi}^*$ -reduce to it.

**Definition B.37.** Given a  $\lambda I$  term  $T$ , the set  $T^{\sim_{\beta\pi}}$  is defined as  $\{U \mid T \sim_{\beta\pi} U \wedge U \subseteq V\}$ .

**Proposition B.38.** If  $U \rightarrow_{\beta\pi} T$  then  $U^{\sim_{\beta\pi}} \subseteq T^{\sim_{\beta\pi}}$ .

**Proposition B.39.** If  $T$  is strongly normalising and  $U \in T^{\sim_{\beta\pi}}$  then  $U$  is strongly normalising.

$$\begin{array}{ccc}
 U \subseteq V & \xrightarrow{\beta\pi} & T \\
 \text{SN} & \xleftarrow{\text{Proposition B.22}} & \text{SN}
 \end{array}$$

*Proof.* By definition,  $U \subseteq V \rightarrow_{\beta\pi}^+ T$ . As  $T$  is strongly normalising,  $V$  is weakly normalising. By Proposition B.22,  $U$  is strongly normalising.  $\square$

**Definition B.40.** The relation  $\mathcal{G}$  between  $\lambda$ -terms and  $\lambda I$ -terms is given by the following rules where  $\vec{t}_k$  denotes the application  $t_1 \dots t_k$ :

$$\begin{array}{c}
 \frac{\forall k \quad t_k \mathcal{G} T_k}{(x \vec{t}_k) \mathcal{G} (x \vec{T}_k)} \mathcal{G}\text{var} \quad \frac{}{((\lambda x.t) t' \vec{t}_k) \mathcal{G} j((\lambda x.t) t' \vec{t}_k)} \mathcal{G}\beta_1 \\
 \\
 \frac{t \mathcal{G} T \quad x \in \text{FV}(T)}{\lambda x.t \mathcal{G} \lambda x.T} \mathcal{G}\lambda \quad \frac{t' \mathcal{G} T' \quad x \notin \text{FV}(t)}{((\lambda x.t) t' \vec{t}_k) \mathcal{G} (j(\lambda x.t) T' j(\vec{t}_k))} \mathcal{G}\beta_2 \\
 \\
 \frac{t \mathcal{G} T \quad N \in \text{SN}_{\beta\pi} \vee N \in T^{\sim_{\beta\pi}}}{t \mathcal{G} [T, N]} \mathcal{G}\text{weak}
 \end{array}$$

Again, we needed to make some changes to the original relation [93] (a proper explanation of the problem is given in our technical report [120]). The non-deterministic rule  $\mathcal{G}\text{weak}$  rule has now been generalised to allow more terms (than Lengrand's relation) to be added to a term. This has implications for the following proofs, most notably Lemma B.41.1 where we weaken the consequent from Lengrand's  $T \in \text{nf}^{\beta\pi}$  to our  $T \in \text{SN}_{\beta\pi}$ . Note that  $\lambda I$  is uniformly normalising (Proposition B.22) which is important, in the following proofs, for our definition of  $\mathcal{G}\text{weak}$ .

**Lemma B.41.**

1. If  $t \in \text{nf}^\beta$  and  $t \mathcal{G} T$ , then  $T \in \text{SN}_{\beta\pi}$ .
2. For any  $\lambda$ -term  $t$ ,  $t \mathcal{G} j(t)$ .

*Proof.* Both proofs induct over the structure of  $t$  where in the first, the  $t = (\lambda x.t')u \vec{t}_k$  case cannot occur as  $t \in \text{nf}^\beta$ .  $\square$

**Definition B.42.** The reduction relation  $\rightsquigarrow$  for  $\lambda$ -terms is defined by the following rules:

$$\begin{array}{c} \frac{t \rightsquigarrow t'}{x \vec{t}_k t \vec{p}_k \rightsquigarrow x \vec{t}_k t' \vec{p}_k} \text{ perp-var} \quad \frac{t \rightsquigarrow t'}{\lambda x.t \rightsquigarrow \lambda x.t'} \text{ perp}\lambda \\[10pt] \frac{x \in \text{FV}(t) \vee t' \in \text{nf}^\beta}{(\lambda x.t) t' \vec{t}_k \rightsquigarrow t\{x \setminus t'\} \vec{t}_k} \text{ perp}\beta_1 \quad \frac{t' \rightsquigarrow t'' \quad x \notin \text{FV}(t)}{(\lambda x.t) t' \vec{t}_k \rightsquigarrow (\lambda x.t) t'' \vec{t}_k} \text{ perp}\beta_2 \end{array}$$

**Theorem B.43.**  $\rightarrow_{\beta\pi}$  strongly simulates  $\rightsquigarrow$  through  $\mathcal{G}$ .

*Proof.* The proof shows, by inducting over the structure of  $u$ , that given the diagram

$$\begin{array}{ccc} \lambda\text{-terms} & & u \rightsquigarrow u' \\ \parallel \mathcal{G} & & \parallel \mathcal{G} \\ \downarrow & & \downarrow \\ \Lambda_I & & U \xrightarrow[\beta\pi]{+} U', \end{array}$$

the dotted arrows may always be filled in. Figure B.2 depicts the various cases.  $\square$

**Corollary B.44.** If  $t \in \text{WN}_{\rightsquigarrow}$  and  $t \mathcal{G} T$  then  $T \in \text{WN}_{\beta\pi}$ .

*Proof.* By induction in  $\text{WN}_{\rightsquigarrow}$ . Letting  $\Lambda$  denote the set of  $\lambda$ -terms, the induction hypothesis is:

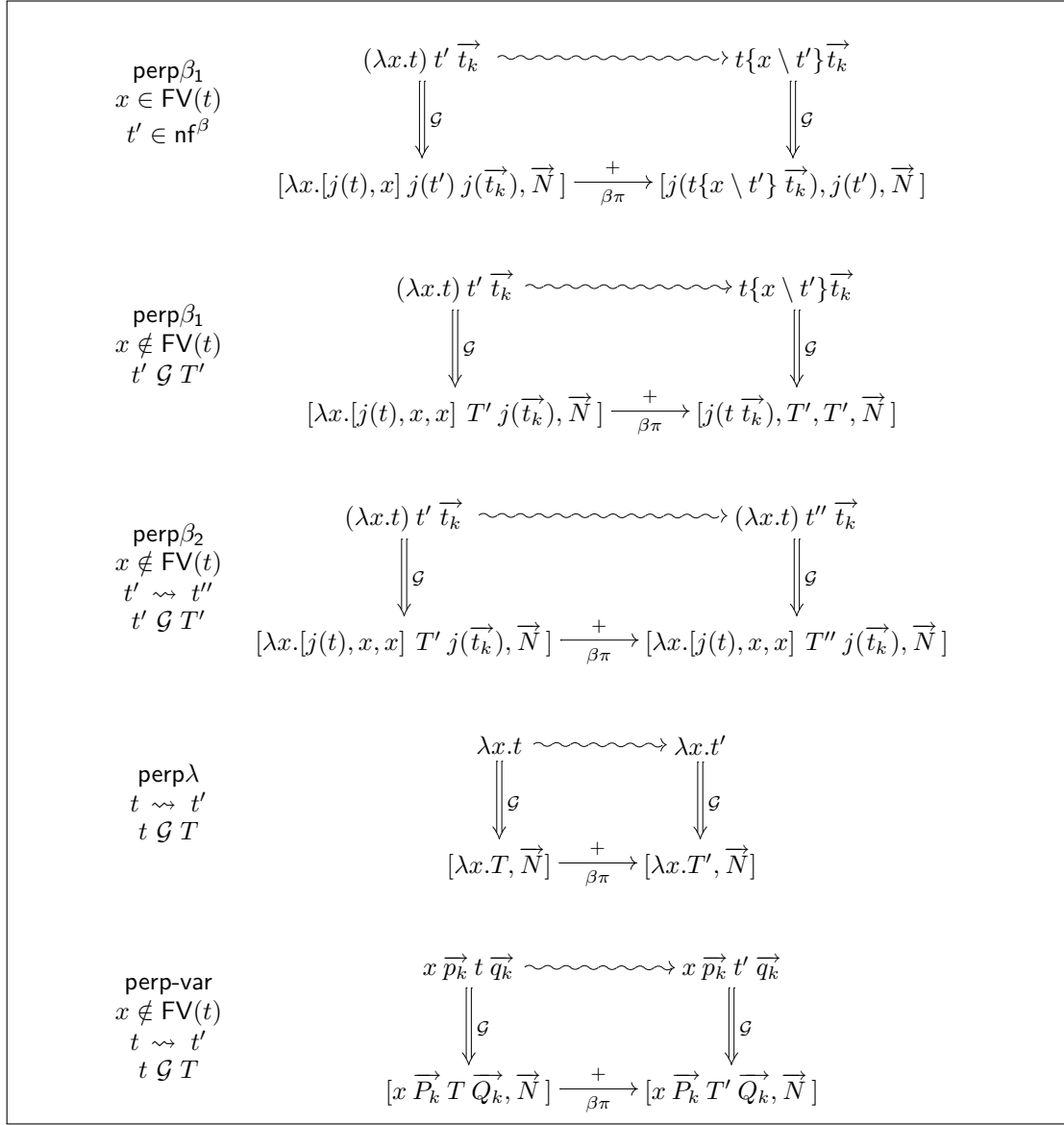
$$(t \in \text{nf}^{\rightsquigarrow}) \vee (\exists u \in \{p \in \Lambda \mid t \rightsquigarrow p\}, \forall U, u \mathcal{G} U \Rightarrow U \in \text{WN}_{\beta\pi})$$

i.e. either  $t$  is in  $\rightsquigarrow$ -normal form or there exists a one-step  $\rightsquigarrow$ -reduct  $u$  of  $t$  such that the proposition holds (all  $\mathcal{G}$ -related terms of  $u$  are  $\rightarrow_{\beta\pi}$ -weakly normalising).

If  $t \in \text{nf}^{\rightsquigarrow}$  then  $T \in \text{SN}_{\beta\pi} \subseteq \text{WN}_{\beta\pi}$  by Lemma B.41.1.

If  $\exists u \in \{p \in \Lambda \mid t \rightsquigarrow p\}, \forall U, u \mathcal{G} U \Rightarrow U \in \text{WN}_{\beta\pi}$ , then Theorem B.43 gives us a specific  $T'$  such that

$$\begin{array}{ccc} t & \rightsquigarrow & u \\ \parallel \mathcal{G} & & \parallel \mathcal{G} \\ \downarrow & & \downarrow \\ T & \xrightarrow[\beta\pi]{+} & T'. \end{array}$$

**Figure B.2:** Strong simulation of  $\rightsquigarrow$  through  $\mathcal{G}$ 

According to the induction hypothesis,  $T' \in \text{WN}_{\beta\pi}$ , and so  $T \in \text{WN}_{\beta\pi}$ . □

**Corollary B.45.**  $j(\text{SN}_\beta) \subseteq \text{WN}_{\beta\pi}$ .

*Proof.*  $\text{SN}_\beta \subseteq \text{SN}_{\rightsquigarrow} \subseteq \text{WN}_{\rightsquigarrow}$ . By Lemma B.41.2,  $\forall t \in \text{SN}_\beta, t \mathcal{G} j(t)$ . By the previous corollary,  $j(t) \in \text{WN}_{\beta\pi}$ . □

**Theorem B.46** (Nederpelt[116]).  $\text{WN}_{\beta\pi} \subseteq \text{SN}_{\beta\pi}$ .

**Corollary B.47.** *For any  $\lambda$ -term  $t$ , if  $t \in \text{SN}_\beta$ , then  $j(t) \in \text{SN}_{\beta\pi}$ .*

*Proof.* By Corollary B.45 and Theorem B.46. □

### B.3.5 Proof of PSN

**Corollary B.48 (PSN).** *For any  $\lambda$ -term  $t$ , if  $t \in \text{SN}_\beta$ , then  $\mathcal{B}(t) \in \text{SN}_{\lambda\text{blxr}}$ .*

*Proof.* If  $t \in \text{SN}_\beta$  then  $j(t) \in \text{SN}_{\beta\pi}$  by Corollary B.47. As  $\mathcal{B}(t) \mathcal{J} j(t)$  by Theorem B.34,  $\mathcal{B}(t) \in \text{SN}_{\lambda\text{blxr}}$  by Corollary B.29. □

### B.3.6 Simplification of the proof

On showing him this work, Stéphane Lengrand had another idea to fix the problem of Section B.3.3 that  $\mathcal{B}(u) \mathcal{J} i(u)$  does not hold. It consisted of changing the  $\mathcal{J}$  relation in the abstraction case and using Klop’s encoding [82, Definition 8.11]:

$$\begin{aligned} \iota(x) &= x \\ \iota(\lambda x.t) &= \lambda x.[\iota(t), x] \\ \iota(tu) &= \iota(t) \iota(u). \end{aligned}$$

This approach may allow use of previous results by Klop to complete the proof and could yield a simpler solution.

### B.3.7 Summary

In summary, we have proved PSN for  $\lambda\text{blxr}$  using the proof for  $\lambda\text{lxr}$  with a few modifications due to the replacement of  $\rightarrow_b$  with  $\rightarrow_{\text{bs}}$ . The modifications we made were:

1. Altering the relation  $\mathcal{I}$  so that  $\rightarrow_{\lambda\text{blxr}}$  was simulated by  $\rightarrow_{\beta\pi}$  through the new relation  $\mathcal{J}$ .
2. Altering the encoding  $i$  so that  $\mathcal{B}(u)$  was related by  $\mathcal{J}$  to the new encoding  $j$ .
3. Proving that  $j$  preserved strong normalisation just as Lengrand proved that  $i$  preserved strong normalisation. This required altering the relation  $\mathcal{G}$  in the  $\mathcal{G}_{\text{weak}}$  case and weakening the remaining propositions.

The alterations to  $\mathcal{I}$  and  $i$  consisted in adding in some redundancy and the alteration to  $\mathcal{G}_{\text{weak}}$  was made to accomodate for this. This redundancy took the form of tacking on (via the “memory operator”) a free variable  $x$  to a  $\lambda I$ -term which already contains a free occurrence of  $x$ . This is analogous to the redundancy in the  $\rightarrow_{\text{bs}}$  rule which creates a second, identical explicit substitution in its firing.



## B.4 Simulation of $\Lambda_{\text{sub}}$ reduction in $\lambda\text{blxr}$

We present the full proof of simulation of  $\Lambda_{\text{sub}}$  reduction in  $\lambda\text{blxr}$  here.  $w$  denotes a  $\Lambda x$ -term below.

**Proposition B.49.** *If  $t \rightarrow_{\text{bcgc}} u$  then  $\mathcal{T}(t)_X \rightarrow_{\lambda\text{blxr}}^+ \mathcal{T}(u)_X$ .*

*Proof.* We write  $[\Theta/\vec{T}]$  to denote a sequence of nested substitutions  $[y_1/T_1] \dots [y_n/T_n]$  and drop the indexing in the translation except for the  $\rightarrow_{\text{gc}}$  case. The proof is by case split, treating the case  $X = \text{FV}(t)$ . Figure 9.1 on page 159 depicts the reduction graphs corresponding to the cases, where  $n$  is the number of copies of the redex generated by the translation.

**Case**  $t \equiv C[(\lambda x.v)w] \rightarrow_b C[v[x/w]] \equiv u, x \in \text{FV}(v)$

Figure B.5 displays the general term<sup>11</sup> for  $\mathcal{T}(t)$  followed by a series of reductions. Figure B.8 displays the general term for  $\mathcal{T}(u)$ . The path  $\mathcal{A}(u) \rightarrow_{\text{p}\downarrow c}^* u'$  involves pushing all substitutions in besides the substitutions which arise from  $[x/w]$ . The renaming  $R_Y^X$  arises from the encoding. The substitutions with binders  $\Phi'$  bind variables in encodings of both  $v$  and  $w$ , the substitutions with binders  $\Xi_v$  and  $\Theta_w$  bind variables of  $v$  or  $w$  respectively. We do not label the context  $D$ ;  $\text{FV}((\lambda x.v)w) = \text{FV}(v[x/w])$  and the same holds for the encodings by Lemma 9.3.1.

We must show that the final terms in both figures are equivalent. This can be shown by proving that  $w_1 \equiv w_1'''$  and  $w_2 \equiv w_2'''$ . These can easily be shown by unwrapping their definitions and renaming variables. For example,

$$\begin{aligned}
 w_2 &\equiv R_{\Psi_1 \Psi_2 \Psi_3 \Psi_4}^{\Pi_1 \Pi_2 \Pi_3 \Pi_4} (\downarrow_{\text{p}\downarrow c} (\downarrow_{\text{p}\downarrow c} (R_{\Pi_1 \Pi' Y}^{\Phi_1 \Phi' X} (\mathcal{A}(w)) [\Theta_w / \vec{S}_w]) [\Pi' / \vec{T}_\Pi])) \\
 &\equiv R_{\Psi_2 \Psi_3}^{\Pi_2 \Pi_3} (\downarrow_{\text{p}\downarrow c} (\downarrow_{\text{p}\downarrow c} (R_{\Psi_1 \Pi' Y}^{\Phi_1 \Phi' \Pi_4 X} (\mathcal{A}(w)) [\Theta_w / \vec{S}_w]) [\Pi' / \vec{T}_\Pi])) \\
 &\equiv R_{\Psi_2 \Psi_3}^{\Pi_2 \Pi_3} (\downarrow_{\text{p}\downarrow c} (\downarrow_{\text{p}\downarrow c} (R_{\Psi_1 \Pi' Y}^{\Phi_1 \Phi' \Pi_4 X} (\mathcal{A}(w)) [\Theta_w / \vec{S}_w]) [\Pi' / R_{\Pi_2}^{\Phi_2} (\vec{T})])) \\
 &\equiv \downarrow_{\text{p}\downarrow c} (\downarrow_{\text{p}\downarrow c} (R_{\Psi_1 \Pi' Y}^{\Phi_1 \Phi' \Pi_4 X} (\mathcal{A}(w)) [\Theta_w / \vec{S}_w]) [\Pi' / \vec{T}_\Psi]) \\
 &\equiv_\alpha \downarrow_{\text{p}\downarrow c} (\downarrow_{\text{p}\downarrow c} (R_{\Psi_1 \Psi' \Theta_\Psi}^{\Phi_1 \Phi' \Theta_w \Pi_4 X} (\mathcal{A}(w)) [\Theta_\Psi / \vec{S}_\Psi]) [\Psi' / \vec{T}_\Psi]) \equiv w_2'''
 \end{aligned}$$

**Case**  $t \equiv C[(\lambda x.v)w] \rightarrow_b C[v[x/w]] \equiv u, x \notin \text{FV}(v)$

This case is similar to the last one. Figure B.11 displays the general term for  $\mathcal{T}(t)$  followed by a series of reductions. Figure B.14 displays the general term for  $\mathcal{T}(u)$ . The renaming  $R_Y^X$  and the substitutions with binders  $\Phi'$ ,  $\Xi_v$ , and  $\Theta_w$  arise as before.

<sup>11</sup>Well, almost. We have omitted some contractions concerning free variables in the substitutions with binders  $\Phi'$ ,  $\Xi_v$ , and  $\Theta_w$  for clarity but Proposition 9.9 can justify this decision.

**Case**  $t \equiv C[C'[x][x/w]] \rightarrow_c C[C'[w][x/w]] \equiv u$ .

We treat the case where the term  $x$  to be replaced is not the only free occurrence of  $x$  in  $C_2[x]$ . The reduction sequences are shown in Figures B.16 and B.17.

The two underlined subterms in those figures reduce to almost the same term – in both, the substitution with binder  $x$  is distributed through the subterms. Both subterms are subterms of the translations  $C_{\Phi}^{\Pi, \Psi_1} C_{\Theta \setminus \Phi}^{\Gamma_2, \Psi_2} (W_{\bar{x}}(\downarrow_{P \downarrow c} (C_{\Pi'}^{\Delta'_1, \Gamma'_1} (D_3[x_i[x_i/v_i]])))[\bar{x}/v_2])$  and  $C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Pi_2, \Psi_2} (W_{\bar{x}}(\downarrow_{P \downarrow c} (C_{\Pi_1}^{\Delta_1, \Gamma_1} C_{\Pi_2}^{\Delta_2, \Gamma_2} (D'_2[v'_i][x/v'_1])))[\bar{x}/v_2])$  of  $\Lambda_{\text{sub}}$  terms. Ignoring linearity and contractions for a moment, the two terms above are identical except that one has a subterm  $v$  where the other has  $x[x/v]$ . By Proposition 9.9, the contractions of both terms are at their most efficient. Therefore, we can write both underlined subterms with the same context  $D$  as  $D[v_i]$  and  $D[x[x/v_i]]$  respectively. The final lines in both sequences then follow and we have  $\mathcal{T}(t) \rightarrow_{\text{Var}}^+ \mathcal{T}(u)$ , one reduction for each copy of the redex  $C_2[x][x/v]$ .

**Case**  $t \equiv C[v[x/w]] \rightarrow_{\text{gc}} C[v] \equiv u, \quad x \notin \text{FV}(v)$ .

Figure B.20 displays the general term for  $\mathcal{T}(t)$  followed by a series of reductions. Figure B.21 displays the general term for  $\mathcal{T}(u)_{\text{FV}(t)}$ .

The terms  $t_1$  and  $\mathcal{T}(u)_{\text{FV}(t)}$  appear very similar. One difference is that  $\mathcal{T}(u)_{\text{FV}(t)}$  has the weakenings  $W_V$  corresponding to the variables lost in the  $\Lambda_{\text{sub}}$  term at top-level. We first pull the weakenings  $\Pi_5$  and their copies (for each copy of the  $\Lambda_{\text{sub}}$  redex induced by the translation) upwards through the term, merging them to reach  $t_2$ .

$t_2$  now closely resembles  $\mathcal{T}(u)_{\text{FV}(t)}$  except for two differences: i)  $t_2$  contains extra weakenings such as  $W_{\Pi_3}$  and  $W_{\Pi_4}$  for each copy of the  $\Lambda_{\text{sub}}$  redex and ii) it also contains contractions involving such sets of variables  $\Pi_3$  and  $\Pi_4$ . We pull all of these weakenings up, merging them with the extra contractions to reach a  $\rightarrow_{\text{Wk}}$  normal form  $t_3$ . Ignoring the placement of contractions,  $t_3$  is now equivalent to  $\mathcal{T}(u)_{\text{FV}(t)}$ .

Finally, we can conclude that this term  $t_3$  is equivalent to  $\mathcal{T}(u)_{\text{FV}(t)}$ : as  $\mathcal{T}(t)$  has its contractions at their most efficient, so does  $t_3$  by Corollary 9.13.

□

$$\begin{aligned}
\text{FV}(R_Y^X(w)) &= \Phi_1 \uplus \Phi' \uplus \Theta_w \uplus \Pi_4 \\
\text{FV}(R_Y^X(v)) &= \Phi_1 \uplus \Phi' \uplus \Xi_v \uplus W \\
\text{FV}(\vec{S}_w) &= \Pi_3 \quad \text{FV}(\vec{T}) = \Phi_2 \quad \vec{T}_\Delta = R_{\Delta_2}^{\Phi_2}(\vec{T}) \quad \vec{T}_\Pi = R_{\Pi_2}^{\Phi_2}(\vec{T})
\end{aligned}$$

**Figure B.3:** Sets of free variables

$$\begin{aligned}
1. \quad v' &= R_{\Delta_1 \Delta' Y}^{\Phi_1 \Phi' X}(\mathcal{A}(v)) \quad w' = R_{\Pi_1 \Pi' Y}^{\Phi_1 \Phi' X}(\mathcal{A}(w)) \\
2. \quad v'' &= \downarrow_{\text{p}\downarrow\text{c}}(v'[\Xi_v/\vec{S}_v]) \quad w'' = \downarrow_{\text{p}\downarrow\text{c}}(w'[\Theta_w/\vec{S}_w]) \\
3. \quad v''' &= \downarrow_{\text{p}\downarrow\text{c}}(v''[\Delta'/\vec{T}_\Delta]) \quad w''' = \downarrow_{\text{p}\downarrow\text{c}}(w''[\Pi'/\vec{T}_\Pi]) \\
4. \quad w_1 &= R_{\Gamma_1 \Gamma_2 \Gamma_3 \Gamma_4}^{\Pi_1 \Pi_2 \Pi_3 \Pi_4}(w''') \quad w_2 = R_{\Psi_1 \Psi_2 \Psi_3 \Psi_4}^{\Pi_1 \Pi_2 \Pi_3 \Pi_4}(w''')
\end{aligned}$$

**Figure B.4:** Abbreviations in Figure B.5

$$\begin{aligned}
&\mathcal{T}(C[(\lambda x.v)w]) \\
&\equiv \downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(C)[R_Y^X(\mathcal{A}((\lambda x.v)w))]) \\
&\equiv_{(1)} D[\downarrow_{\text{p}\downarrow\text{c}}(C_{\Phi_1}^{\Delta_1, \Pi_1} C_{\Phi'}^{\Delta', \Pi'}((\lambda x.v')w')[\Phi'/\vec{T}][\Xi_v/\vec{S}_v][\Theta_w/\vec{S}_w])] \\
&\equiv_{(2)} D[\downarrow_{\text{p}\downarrow\text{c}}(C_{\Phi_1}^{\Delta_1, \Pi_1} C_{\Phi'}^{\Delta', \Pi'}((\lambda x.v'')w'')[\Phi'/\vec{T}])] \\
&\equiv D[C_{\Phi_1}^{\Delta_1, \Pi_1} C_{\Phi_2}^{\Delta_2, \Pi_2}((\lambda x. \downarrow_{\text{p}\downarrow\text{c}}(v''[\Delta'/\vec{T}_\Delta])) \downarrow_{\text{p}\downarrow\text{c}}(w''[\Pi'/\vec{T}_\Pi]))] \\
&\equiv_{(3)} D[C_{\Phi_1 \Phi_2}^{\Delta_1 \Delta_2, \Pi_1 \Pi_2}((\lambda x.v''')w''')] \\
&\rightarrow_{\text{bs}} (4) D[C_{\Phi_1 \Phi_2}^{\Delta_1 \Delta_2, \Pi_1 \Pi_2} C_{\Pi_1 \Pi_2}^{\Gamma_1 \Gamma_2, \Psi_1 \Psi_2} C_{\Pi_3 \Pi_4}^{\Gamma_3 \Gamma_4, \Psi_3 \Psi_4}(W_{\bar{x}}(v'''[x/w_1])[\bar{x}/w_2])] \\
&\rightarrow_{\text{Ctn}} D[C_{\Phi_1 \Phi_2}^{\Pi_1 \Pi_2, \Psi_1 \Psi_2} C_{\Pi_3 \Pi_4}^{\Gamma_3 \Gamma_4, \Psi_3 \Psi_4}(W_{\bar{x}}(C_{\Pi_1 \Pi_2}^{\Delta_1 \Delta_2, \Gamma_1 \Gamma_2}(v'''[x/w_1])[\bar{x}/w_2])] \\
&\rightarrow_{\text{p}\downarrow\text{c}}^+ D[C_{\Phi_1 \Phi_2}^{\Pi_1 \Pi_2, \Psi_1 \Psi_2} C_{\Pi_3 \Pi_4}^{\Gamma_3 \Gamma_4, \Psi_3 \Psi_4}(W_{\bar{x}}(\downarrow_{\text{p}\downarrow\text{c}}(C_{\Pi_1 \Pi_2}^{\Delta_1 \Delta_2, \Gamma_1 \Gamma_2}(v'''[x/w_1]))[\bar{x}/w_2])]
\end{aligned}$$

**Figure B.5:** Translation for  $C[(\lambda x.v)w]$ ,  $x$  a free variable of  $v$

$$\begin{aligned}
\text{FV}(R_Y^X(w)) &= \Phi_1 \uplus \Phi' \uplus \Theta_w \uplus \Pi_4 \\
\text{FV}(R_Y^X(v)) &= \Phi_1 \uplus \Phi' \uplus \Xi_v \uplus W \\
\text{FV}(\vec{S}_w) &= \Pi_3 \quad \vec{S}_\Gamma = R_{\Gamma_3}^{\Pi_3}(\vec{S}_w) \quad \vec{S}_\Psi = R_{\Psi_3}^{\Pi_3}(\vec{S}_w) \\
\text{FV}(\vec{T}) &= \Phi_2 \quad \vec{T}_\Pi = R_{\Pi_2}^{\Phi_2}(\vec{T}) \quad \vec{T}_\Psi = R_{\Psi_2}^{\Phi_2}(\vec{T}) \quad \vec{T}_\Gamma = R_{\Gamma_2}^{\Phi_2}(\vec{T}) \quad \vec{T}_\Delta = R_{\Delta_2}^{\Phi_2}(\vec{T})
\end{aligned}$$

**Figure B.6:** Sets of free variables

1.  $v' = R_{\Delta_1 \Delta' Y}^{\Phi_1 \Phi' X}(\mathcal{A}(v)) \quad w'_1 = R_{\Gamma_1 \Gamma' \Theta_\Gamma \Gamma_4 Y}^{\Phi_1 \Phi' \Theta_w \Pi_4 X}(\mathcal{A}(w)) \quad w'_2 = R_{\Psi_1 \Psi' \Theta_\Psi \Psi_4 Y}^{\Phi_1 \Phi' \Theta_w \Pi_4 X}(\mathcal{A}(w))$
2.  $v'' = \downarrow_{\text{p}\downarrow\text{c}}(v'[\Xi_v/\vec{S}_v])$
3.  $w'_1 = \downarrow_{\text{p}\downarrow\text{c}}(w'_1[\Theta_\Gamma/\vec{S}_\Gamma]) \quad w'_2 = \downarrow_{\text{p}\downarrow\text{c}}(w'_2[\Theta_\Psi/\vec{S}_\Psi])$
4.  $v''' = \downarrow_{\text{p}\downarrow\text{c}}(v''[\Delta'/\vec{T}_\Delta]) \quad w'_1 = \downarrow_{\text{p}\downarrow\text{c}}(w'_1[\Gamma'/\vec{T}_\Gamma]) \quad w'_2 = \downarrow_{\text{p}\downarrow\text{c}}(w'_2[\Psi'/\vec{T}_\Psi])$

**Figure B.7:** Abbreviations in Figure B.8

$$\begin{aligned}
&\mathcal{T}(C[v[x/w]]) \\
&\equiv \downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(C)[R_Y^X(\mathcal{A}(v[x/w]))]) \\
&\equiv(1) D[\downarrow_{\text{p}\downarrow\text{c}}(C_{\Phi_1 \Phi'}^{\Pi_1 \Pi', \Psi_1 \Psi'} C_{\Theta_w \Pi_4}^{\Theta_\Gamma \Gamma_4, \Theta_\Psi \Psi_4}(W_{\bar{x}}(C_{\Pi_1 \Pi'}^{\Delta_1 \Delta', \Gamma_1 \Gamma'}(v'[x/w'_1]))[\bar{x}/w'_2]))[\Phi'/\vec{T}][\Xi_v/\vec{S}_v][\Theta_w/\vec{S}_w])] \\
&\equiv(2) D[\downarrow_{\text{p}\downarrow\text{c}}(C_{\Phi_1 \Phi'}^{\Pi_1 \Pi', \Psi_1 \Psi'} C_{\Pi_3 \Pi_4}^{\Gamma_3 \Gamma_4, \Psi_3 \Psi_4}(W_{\bar{x}}(C_{\Pi_1 \Pi'}^{\Delta_1 \Delta', \Gamma_1 \Gamma'}(v''[x/w'_1[\Theta_\Gamma/\vec{S}_\Gamma]]))[\bar{x}/w'_2[\Theta_\Psi/\vec{S}_\Psi]]))[\Phi'/\vec{T}])] \\
&\equiv(3) D[\downarrow_{\text{p}\downarrow\text{c}}(C_{\Phi_1 \Phi'}^{\Pi_1 \Pi', \Psi_1 \Psi'} C_{\Pi_3 \Pi_4}^{\Gamma_3 \Gamma_4, \Psi_3 \Psi_4}(W_{\bar{x}}(C_{\Pi_1 \Pi'}^{\Delta_1 \Delta', \Gamma_1 \Gamma'}(v''[x/w'_1]))[\bar{x}/w'_2]))[\Phi'/\vec{T}])] \\
&\equiv D[\downarrow_{\text{p}\downarrow\text{c}}(C_{\Phi_1 \Phi_2}^{\Pi_1 \Pi_2, \Psi_1 \Psi_2} C_{\Pi_3 \Pi_4}^{\Gamma_3 \Gamma_4, \Psi_3 \Psi_4}(W_{\bar{x}}(C_{\Pi_1 \Pi'}^{\Delta_1 \Delta', \Gamma_1 \Gamma'}(v''[x/w'_1]))[\Pi'/\vec{T}_\Pi])[\bar{x}/w'_2[\Psi'/\vec{T}_\Psi]]))] \\
&\equiv D[\downarrow_{\text{p}\downarrow\text{c}}(C_{\Phi_1 \Phi_2}^{\Pi_1 \Pi_2, \Psi_1 \Psi_2} C_{\Pi_3 \Pi_4}^{\Gamma_3 \Gamma_4, \Psi_3 \Psi_4}(W_{\bar{x}}(C_{\Pi_1 \Pi_2}^{\Delta_1 \Delta_2, \Gamma_1 \Gamma_2}(\downarrow_{\text{p}\downarrow\text{c}}(v''[\Delta'/\vec{T}_\Delta])[x/\downarrow_{\text{p}\downarrow\text{c}}(w'_1[\Gamma'/\vec{T}_\Gamma]]))[\bar{x}/\downarrow_{\text{p}\downarrow\text{c}}(w'_2[\Psi'/\vec{T}_\Psi]])))] \\
&\equiv(4) D[C_{\Phi_1 \Phi_2}^{\Pi_1 \Pi_2, \Psi_1 \Psi_2} C_{\Pi_3 \Pi_4}^{\Gamma_3 \Gamma_4, \Psi_3 \Psi_4}(W_{\bar{x}}(\downarrow_{\text{p}\downarrow\text{c}}(C_{\Pi_1 \Pi_2}^{\Delta_1 \Delta_2, \Gamma_1 \Gamma_2}(v'''[x/w'_1''']))[\bar{x}/w'_2''']))]
\end{aligned}$$

**Figure B.8:** Translation for  $C[v[x/w]]$ ,  $x$  a free variable of  $v$

$$\begin{aligned}\text{FV}(R_Y^X(w)) &= \Phi_1 \uplus \Phi' \uplus \Theta_w \uplus \Pi_4 \\ \text{FV}(R_Y^X(v)) &= \Phi_1 \uplus \Phi' \uplus \Xi_v \uplus W \\ \text{FV}(\vec{S}_w) &= \Pi_3 \quad \text{FV}(\vec{T}) = \Phi_2 \quad \vec{T}_\Delta = R_{\Delta_2}^{\Phi_2}(\vec{T}) \quad \vec{T}_\Pi = R_{\Pi_2}^{\Phi_2}(\vec{T})\end{aligned}$$

**Figure B.9:** Sets of free variables

$$\begin{aligned}1. \quad v' &= R_{\Delta_1 \Delta' Y}^{\Phi_1 \Phi' X}(\mathcal{A}(v)) \quad w' = R_{\Pi_1 \Pi' Y}^{\Phi_1 \Phi' X}(\mathcal{A}(w)) \\ 2. \quad v'' &= \downarrow_{\text{p}\downarrow\text{c}}(v'[\Xi_v/\vec{S}_v]) \quad w'' = \downarrow_{\text{p}\downarrow\text{c}}(w'[\Theta_w/\vec{S}_w]) \\ 3. \quad v_1 &= \downarrow_{\text{p}\downarrow\text{c}}(v''[\Delta'/\vec{T}_\Delta]) \quad w_1 = \downarrow_{\text{p}\downarrow\text{c}}(w''[\Pi'/\vec{T}_\Pi])\end{aligned}$$

**Figure B.10:** Abbreviations in Figure B.11

$$\begin{aligned}& \mathcal{T}(C[(\lambda x.v)w]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(C[R_Y^X(\mathcal{A}((\lambda x.v)w))]) \\ \equiv_{(1)} & D[\downarrow_{\text{p}\downarrow\text{c}}((C_{\Phi_1 \Phi'}^{\Delta_1 \Delta', \Pi_1 \Pi'}((\lambda x.W_x(v'))w'))[\Phi'/\vec{T}][\Xi_v/\vec{S}_v][\Theta_w/\vec{S}_w])] \\ \equiv_{(2)} & D[\downarrow_{\text{p}\downarrow\text{c}}((C_{\Phi_1 \Phi'}^{\Delta_1 \Delta', \Pi_1 \Pi'}((\lambda x.W_x(v''))w''))[\Phi'/\vec{T}])] \\ \equiv & D[C_{\Phi_1 \Phi_2}^{\Delta_1 \Delta_2, \Pi_1 \Pi_2}((\lambda x. \downarrow_{\text{p}\downarrow\text{c}}(W_x(v''[\Delta'/\vec{T}_\Delta]))) \downarrow_{\text{p}\downarrow\text{c}}(w''[\Pi'/\vec{T}_\Pi]))] \\ \equiv_{(3)} & D[C_{\Phi_1 \Phi_2}^{\Delta_1 \Delta_2, \Pi_1 \Pi_2}((\lambda x.W_x(v_1))w_1)] \\ \rightarrow_{\text{bs}} & D[C_{\Phi_1 \Phi_2}^{\Delta_1 \Delta_2, \Pi_1 \Pi_2}(C_{\text{FV}(w_1)}^{\Gamma, \Psi}(W_{\bar{x}}(W_x(v_1)[x/R_{\Gamma}^{\text{FV}(w_1)}(w_1)])[\bar{x}/R_{\Psi}^{\text{FV}(w_1)}(w_1)]))] \\ \rightarrow_{\text{Weak}_1} & D[C_{\Phi_1 \Phi_2}^{\Delta_1 \Delta_2, \Pi_1 \Pi_2}(C_{\text{FV}(w_1)}^{\Gamma, \Psi}(W_{\Psi}(W_x(v_1)[x/R_{\Gamma}^{\text{FV}(w_1)}(w_1)])))] \\ \rightarrow_{\text{Merge}} & D[C_{\Phi_1 \Phi_2}^{\Delta_1 \Delta_2, \Pi_1 \Pi_2}(W_x(v_1)[x/w_1])]\end{aligned}$$

**Figure B.11:** Translation for  $C[v[x/w]]$ ,  $x$  not a free variable of  $v$

$$\begin{aligned}
\text{FV}(R_Y^X(w)) &= \Phi_1 \uplus \Phi' \uplus \Theta_w \uplus \Pi_4 \\
\text{FV}(R_Y^X(v)) &= \Phi_1 \uplus \Phi' \uplus \Xi_v \uplus W \\
\text{FV}(\vec{T}) &= \Phi_2 \quad \vec{T}_\Delta = R_{\Delta_2}^{\Phi_2}(\vec{T}) \quad \vec{T}_\Pi = R_{\Pi_2}^{\Phi_2}(\vec{T})
\end{aligned}$$

**Figure B.12:** Sets of free variables

1.  $v' = R_{\Delta_1 \Delta' Y}^{\Phi_1 \Phi' X}(\mathcal{A}(v)) \quad w' = R_{\Pi_1 \Pi' Y}^{\Phi_1 \Phi' X}(\mathcal{A}(w))$
2.  $v'' = \downarrow_{\text{p}\downarrow\text{c}}(v'[\Xi_v/\vec{S}_v]) \quad w'' = \downarrow_{\text{p}\downarrow\text{c}}(w'[\Theta_w/\vec{S}_w])$
3.  $v_1 = \downarrow_{\text{p}\downarrow\text{c}}(v''[\Delta'/\vec{T}_\Delta]) \quad w_1 = \downarrow_{\text{p}\downarrow\text{c}}(w''[\Pi'/\vec{T}_\Pi])$

**Figure B.13:** Abbreviations in Figure B.14

$$\begin{aligned}
&\mathcal{T}(C[v[x/w]]) \\
&\equiv \downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(C)[R_Y^X(\mathcal{A}(v[x/w]))]) \\
&\equiv_{(1)} D[\downarrow_{\text{p}\downarrow\text{c}}((C_{\Phi_1 \Phi'}^{\Delta_1 \Delta', \Pi_1 \Pi'}(W_x(v')[x/w'])(\Phi'/\vec{T})[\Xi_v/\vec{S}_v][\Theta_w/\vec{S}_w]))] \\
&\equiv_{(2)} D[\downarrow_{\text{p}\downarrow\text{c}}((C_{\Phi_1 \Phi'}^{\Delta_1 \Delta', \Pi_1 \Pi'}(W_x(v'')[x/w''])(\Phi'/\vec{T})))] \\
&\equiv D[(C_{\Phi_1 \Phi_2}^{\Delta_1 \Delta_2, \Pi_1 \Pi_2}(W_x(\downarrow_{\text{p}\downarrow\text{c}}(v''[\Delta'/\vec{T}_\Delta]))[x/\downarrow_{\text{p}\downarrow\text{c}}(w''[\Pi'/\vec{T}_\Pi])]))] \\
&\equiv_{(3)} D[C_{\Phi_1 \Phi_2}^{\Delta_1 \Delta_2, \Pi_1 \Pi_2}(W_x(v_1)[x/w_1])]
\end{aligned}$$

**Figure B.14:** Translation for  $C[v[x/w]]$ ,  $x$  not a free variable of  $v$

$$\text{FV}(v) = \Theta \quad v_2 = R_{\Psi_1 \Psi_2}^{\Phi \ \Theta \setminus \Phi}(\mathcal{A}(v))$$

**Figure B.15:** Free variables and abbreviations for Figures B.16 and B.17

$$\begin{aligned} & \mathcal{T}(C_1[C_2[x][x/v]]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(C_1)[R_X^W(\mathcal{A}(C_2[x][x/v]))]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W((C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Gamma_2, \Psi_2}(W_{\bar{x}}(C_{\Pi_1}^{\Delta_1, \Gamma_1}(D_2[x_i][x/v_1]))[\bar{x}/v_2]))])]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W(C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Gamma_2, \Psi_2}(W_{\bar{x}}(C_{\Pi_1'}^{\Delta_1', \Gamma_1'}(D_3[x_i[x_i/v_i]]))[\bar{x}/v_2]))])]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W(C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Gamma_2, \Psi_2}(W_{\bar{x}}(\downarrow_{\text{p}\downarrow\text{c}}(C_{\Pi'}^{\Delta_1', \Gamma_1'}(D_3[x_i[x_i/v_i]]))[\bar{x}/v_2]))])])]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W(C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Gamma_2, \Psi_2}(W_{\bar{x}}(D[x_i[x_i/v_i]])[\bar{x}/v_2]))]) \quad \equiv \quad E[x_i[x_i/v_i]] \end{aligned}$$

**Figure B.16:** Translation for  $C_1[C_2[x][x/v]]$

$$\begin{aligned} & \mathcal{T}(C_1[C_2'[v][x/v]]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(C_1)[R_X^W(\mathcal{A}(C_2'[v][x/v]))]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W(C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Pi_2, \Psi_2}(W_{\bar{x}}(C_{\Pi_1}^{\Delta_1, \Gamma_1} C_{\Pi_2}^{\Delta_2, \Gamma_2}(D_2'[v_i][x/v_1']))[\bar{x}/v_2]))])]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W(C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Pi_2, \Psi_2}(W_{\bar{x}}(\downarrow_{\text{p}\downarrow\text{c}}(C_{\Pi_1}^{\Delta_1, \Gamma_1} C_{\Pi_2}^{\Delta_2, \Gamma_2}(D_2'[v_i][x/v_1'])))[\bar{x}/v_2]))])])]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W(C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Pi_2, \Psi_2}(W_{\bar{x}}(D[v_i])[\bar{x}/v_2]))]) \quad \equiv \quad E[v_i] \end{aligned}$$

**Figure B.17:** Translation for  $C_1[C_2[v][x/v]]$ ,  $x$  a free variable of  $C_2[v]$

$$\begin{aligned} & \mathcal{T}(C_1[C_2'[v][x/v]]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(C_1)[R_X^W(\mathcal{A}(C_2'[v][x/v]))]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W(C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Pi_2, \Psi_2}(W_x(D_2'[v_i'])[x/v_2]))])]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W(C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Pi_2, \Psi_2}(W_x(\downarrow_{\text{p}\downarrow\text{c}}(D_2'[v_i'])))[x/v_2]))])]) \\ \equiv & \downarrow_{\text{p}\downarrow\text{c}}(D_1[R_X^W(C_{\Phi}^{\Pi_1, \Psi_1} C_{\Theta \setminus \Phi}^{\Pi_2, \Psi_2}(W_x(D[v_i]))[x/v_2]))]) \quad \equiv \quad E[v_i] \end{aligned}$$

**Figure B.18:** Translation for  $C_1[C_2[v][x/v]]$ ,  $x$  not a free variable of  $C_2[v]$

$$\begin{aligned}
W &\subseteq \text{FV}(v) \quad Y \subseteq \text{FV}(w) \setminus \text{FV}(v) \\
\text{FV}(R_{XZ}^{WY}(v)) &= \text{FV}(R_X^W(v)) = \Phi_1 \uplus \Phi' \uplus \Xi_v \uplus W \\
\text{FV}(R_{XZ}^{WY}(w)) &= \Phi_1 \uplus \Phi' \uplus \Theta_w \uplus \Pi_4 \uplus \Pi_5 \\
Y &\subseteq \Theta_w \uplus \Pi_4 \uplus \Pi_5 \\
D[\ ] &= W_{U \setminus \text{FV}(t)}[\mathcal{T}(C[\ ])]_{\emptyset} \\
V &\text{ is the set of free variables of } w \text{ not bound in } t \text{ or not occurring free above the hole in } C[\ ] \text{ or in } v \text{ (the lost variables)} \\
D'[\ ] &= W_V D_2[\ ] \\
C'[\ ] &= C[\ ] \text{ except that the free variables of the hole in } C'[\ ] \text{ do not contain } V \\
\Pi_5 &= R_Z^Y(V) \\
\text{FV}(\vec{S}_w) &= \Pi_3 \quad \text{FV}(\vec{T}) = \Phi_2 \quad \vec{T}_\Delta = R_{\Delta_2}^{\Phi_2}(\vec{T}) \quad \vec{T}_\Pi = R_{\Pi_2}^{\Phi_2}(\vec{T})
\end{aligned}$$

**Figure B.19:** Free variables and abbreviations for Figures B.20 and B.21

$$\begin{aligned}
&\mathcal{T}(C[v[x/w]])_U \\
\equiv &\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(C)_U[R_{XZ}^{WY}(\mathcal{A}(v[x/w]))]) \\
\equiv &D[\downarrow_{\text{p}\downarrow\text{c}}((C_{\Phi_1\Phi'}^{\Delta_1\Delta',\Pi_1\Pi'}(W_x(\mathcal{A}(R_{\Delta_1\Delta'}^{\Phi_1\Phi'}(v)))[x/\mathcal{A}(R_{\Pi_1\Pi'}^{\Phi_1\Phi'}(w))]))[\Phi'/\vec{T}][\Xi_v/\vec{S}_v][\Theta_w/\vec{S}_w])] \\
\equiv &D[C_{\Phi_1\Phi_2}^{\Delta_1\Delta_2,\Pi_1\Pi_2}(W_x(\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(R_{\Delta_1\Delta'}^{\Phi_1\Phi'}(v))[\Delta'/\vec{T}_\Delta][\Xi_v/\vec{S}_v]))[x/\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(R_{\Pi_1\Pi'}^{\Phi_1\Phi'}(w))[\Pi'/\vec{T}_\Pi][\Theta_w/\vec{S}_w])))] \\
\rightarrow_{\text{Weak}_1} &D[C_{\Phi_1\Phi_2}^{\Delta_1\Delta_2,\Pi_1\Pi_2}(W_{\Pi_1\Pi_2\Pi_3\Pi_4\Pi_5}(\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(R_{\Delta_1\Delta'}^{\Phi_1\Phi'}(v))[\Delta'/\vec{T}_\Delta][\Xi_v/\vec{S}_v])))] \\
\rightarrow_{\text{Merge}} &D[R_{\Phi_1\Phi_2}^{\Delta_1\Delta_2}(W_{\Pi_3\Pi_4\Pi_5}(\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(R_{\Delta_1\Delta'}^{\Phi_1\Phi'}(v))[\Delta'/\vec{T}_\Delta][\Xi_v/\vec{S}_v])))] \\
\equiv &D[W_{\Pi_3\Pi_4\Pi_5}(\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(R_{\Delta'}^{\Phi'}(v))[\Delta'/\vec{T}][\Xi_v/\vec{S}_v]))] \\
\equiv_{\alpha} &D[W_{\Pi_3\Pi_4\Pi_5}(\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(R_X^W(v))[\Phi'/\vec{T}][\Xi_v/\vec{S}_v]))] \\
\equiv &t_1 \rightarrow_{\text{Wk}} W_V D_1[W_{\Pi_3\Pi_4}(\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(R_X^W(v))[\Phi'/\vec{T}][\Xi_v/\vec{S}_v]))] \equiv t_2 \rightarrow_{\text{Wk}} t_3
\end{aligned}$$

**Figure B.20:** Translation for  $C[v[x/w]]$ ,  $x$  not a free variable of  $v$ 

$$\begin{aligned}
&\mathcal{T}(C'[v])_U \\
\equiv &\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(C)_U[R_X^W(\mathcal{A}(v))]) \\
\equiv &D'[\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(R_X^W(v))[\Phi'/\vec{T}][\Xi_v/\vec{S}_v])] \\
\equiv &W_V D_2[\downarrow_{\text{p}\downarrow\text{c}}(\mathcal{A}(R_X^W(v))[\Phi'/\vec{T}][\Xi_v/\vec{S}_v])]
\end{aligned}$$

**Figure B.21:** Translation for  $C[v]$



## B.5 Simulation of $\Lambda_{\text{sub}}$ reduction in $\lambda\text{es}$

**Proposition B.50** ( $\lambda\text{es}$  simulates  $\Lambda_{\text{sub}}$  through  $\mathcal{T}$ ). *If  $t \rightarrow_{\Lambda_{\text{sub}}} t'$  then  $\mathcal{T}(t) \rightarrow_{\lambda\text{es}}^+ \mathcal{T}(t')$*

*Proof.* We break the proof over the possible reductions at the root, recalling that translations of terms are in  $\text{ALC}$ -normal form.

- $(\lambda x.u) v \rightarrow_{\text{B}} u[x/v]$ .

$$\begin{array}{ll}
 x \in \text{fv}(u) & x \notin \text{fv}(u) \\
 \mathcal{T}((\lambda x.u) v) & \mathcal{T}((\lambda x.u) v) \\
 = ((\lambda x.\mathcal{T}(u)) \mathcal{T}(v))[z/\mathcal{T}(v)] & = ((\lambda x.\mathcal{T}(u)) \mathcal{T}(v))[z/\mathcal{T}(v)] \\
 \rightarrow_{\text{B}} \mathcal{T}(u)[x/\mathcal{T}(v)][z/\mathcal{T}(v)] & \rightarrow_{\text{B}} \mathcal{T}(u)[x/\mathcal{T}(v)][z/\mathcal{T}(v)] \\
 \rightarrow_{\text{ALC}} \text{ALC}(\mathcal{T}(u)[x/\mathcal{T}(v)][z/\mathcal{T}(v)]) & \rightarrow_{\text{gc}} \mathcal{T}(u)[x/\mathcal{T}(v)] \\
 = \mathcal{T}(u[x/v]) & \rightarrow_{\text{ALC}} \text{ALC}(\mathcal{T}(u)[x/\mathcal{T}(v)]) \\
 & = \mathcal{T}(u[x/v]).
 \end{array}$$

- $C[x][x/u] \rightarrow_{\text{c}} C[u][x/u]$ . We induct over the structure of  $C$ . All cases are covered in the technical report [81].

–  $C = D v$ .

We treat the case where  $x \notin \text{fv}(v)$ . By i.h.,  $\mathcal{T}(D[x][x/u]) = \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) \rightarrow_{\lambda\text{es}}^+ \mathcal{T}(D[u][x/u])$ . We have

$$\begin{aligned}
 \mathcal{T}(D[u][x/u]) &= \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)]) & x \notin \text{fv}(D[u]) \\
 \mathcal{T}(D[u][x/u]) &= \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) & x \in \text{fv}(D[u])
 \end{aligned}$$

so by Lemma 9.27,

$$\begin{aligned}
 \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)]) &\rightarrow_{\lambda\text{es}}^+ \text{ALC}(\mathcal{T}(D[u])) & x \notin \text{fv}(D[u]) \\
 \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)]) &\rightarrow_{\lambda\text{es}}^+ \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)]) & x \in \text{fv}(D[u]).
 \end{aligned}$$

When  $x \in \text{fv}(D[u])$  then

$$\begin{aligned}
 &\mathcal{T}((D[x] v)[x/u]) \\
 &= \text{ALC}((\mathcal{T}(D[x]) \mathcal{T}(v))[z/\mathcal{T}(v)][x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) \\
 &= (\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)]) \text{ALC}(\mathcal{T}(v)))[z/\mathcal{T}(v)][\bar{x}/\mathcal{T}(u)] \\
 &\rightarrow_{\lambda\text{es}}^+ (\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)]) \text{ALC}(\mathcal{T}(v)))[z/\mathcal{T}(v)][\bar{x}/\mathcal{T}(u)] \\
 &= (\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)] \mathcal{T}(v))[z/\mathcal{T}(v)][\bar{x}/\mathcal{T}(u)] \\
 &= \text{ALC}((\mathcal{T}(D[u]) \mathcal{T}(v))[z/\mathcal{T}(v)][x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) \\
 &= \mathcal{T}((D[u] v)[x/u]).
 \end{aligned}$$

The case where  $x \notin \text{fv}(D[u])$  follows similarly as do the cases where  $x \in \text{fv}(v)$ .

- $C = v D$ . Similar.
- $C = \lambda z.D$ . Similar.
- $C = D[z/v]$ .

The case is further broken down depending on whether  $z \in \text{fv}(D[x])$ ,  $x \in \text{fv}(D[u])$ , and  $x \in \text{fv}(v)$ .

$\mathcal{T}(D[x][x/u]) = \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) \rightarrow_{\lambda\text{es}}^+ \mathcal{T}(D[u][x/u])$  by i.h.. We have

$$\begin{aligned} \mathcal{T}(D[u][x/u]) &= \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)]) & x \notin \text{fv}(D[u]) \\ \mathcal{T}(D[u][x/u]) &= \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) & x \in \text{fv}(D[u]) \end{aligned}$$

so by Lemma 9.27,

$$\begin{aligned} \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)]) &\rightarrow_{\lambda\text{es}}^+ \text{ALC}(\mathcal{T}(D[u])) & x \notin \text{fv}(D[u]) \\ \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)]) &\rightarrow_{\lambda\text{es}}^+ \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)]) & x \in \text{fv}(D[u]) \end{aligned}$$

As  $z \in \text{fv}(D[x])$  and  $z \neq x$  implies  $z \in \text{fv}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])$  and  $z \in \text{fv}(\mathcal{T}(D[u]))$ , then by Lemma 9.24.1 and 9.24.2 we have

$$\begin{aligned} \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)][z/v']) &\rightarrow_{\lambda\text{es}}^+ \text{ALC}(\mathcal{T}(D[u])[z/v']) & x \notin \text{fv}(D[u]) \\ \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)][z/v']) &\rightarrow_{\lambda\text{es}}^+ \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)][z/v']) & x \in \text{fv}(D[u]). \end{aligned}$$

We take the case where  $x \in \text{fv}(D[u])$ ,  $z \in \text{fv}(D[x])$ ,  $x \notin \text{fv}(v)$ . We assume below that  $D[x] \neq \lambda y.s$  for some  $s$  – the alternative case can be similarly proved.

$$\begin{aligned} &\mathcal{T}(C[x][x/u]) \\ &= \text{ALC}(\mathcal{T}(D[x])[z/\mathcal{T}(v)][\bar{z}/\mathcal{T}(v)][x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) \\ &= \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)][z/\mathcal{T}(v)][\bar{z}/\mathcal{T}(v)][\bar{x}/\mathcal{T}(u)]) \\ &\rightarrow_{\lambda\text{es}}^+ \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)][z/\mathcal{T}(v)][\bar{z}/\mathcal{T}(v)][\bar{x}/\mathcal{T}(u)]) \\ &= \text{ALC}(\mathcal{T}(D[u])[z/\mathcal{T}(v)][\bar{z}/\mathcal{T}(v)][x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) \\ &= \mathcal{T}(C[u][x/u]) \end{aligned}$$

- $C = v[z/D]$ .

$\mathcal{T}(D[x][x/u]) = \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) \rightarrow_{\lambda\text{es}}^+ \mathcal{T}(D[u][x/u])$  by i.h.. We assume below that  $v \neq \lambda y.s$  for some  $s$  – the alternative is similarly explained.

The case is further broken down depending on whether  $z \in \text{fv}(v)$ ,  $x \in \text{fv}(D[u])$ , and  $x \in \text{fv}(v)$ . For the cases where  $x \notin \text{fv}(D[u])$ , we use the equations and reductions from Figure B.22. For the cases where  $x \in \text{fv}(D[u])$ , we use the equations and reductions from Figure B.23.

The structure of the proofs is the same. Depending on  $z \in \text{fv}(v)$ ,  $x \notin \text{fv}(v)$ , or  $x \in \text{fv}(D[u])$ , we use a different equation from the figures as follows:

|                                                     | $x \notin \text{fv}(D[u])$ | $x \in \text{fv}(D[u])$ |
|-----------------------------------------------------|----------------------------|-------------------------|
| $z \notin \text{fv}(v)$ and $x \notin \text{fv}(v)$ | Use (B.1)                  | Use (B.5)               |
| $z \notin \text{fv}(v)$ and $x \in \text{fv}(v)$    | Use (B.2)                  | Use (B.6)               |
| $z \in \text{fv}(v)$ and $x \notin \text{fv}(v)$    | Use (B.3)                  | Use (B.7)               |
| $z \in \text{fv}(v)$ and $x \in \text{fv}(v)$       | Use (B.4)                  | Use (B.8)               |

For example, we take the case where  $z \in \text{fv}(v)$ ,  $x \notin \text{fv}(v)$ , and  $x \in \text{fv}(D[u])$ .

$$\begin{aligned}
& \mathcal{T}(C[x][x/u]) \\
&= \text{ALC}(\mathcal{T}(v)[z/\mathcal{T}(D[x])][\bar{z}/\mathcal{T}(D[x])][x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) \\
&= \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{x}/\mathcal{T}(u)]) \\
&\xrightarrow{+}_{\lambda\text{es}} \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])][\bar{x}/\mathcal{T}(u)]) \\
&= \text{ALC}(\mathcal{T}(v)[z/\mathcal{T}(D[u])][\bar{z}/\mathcal{T}(D[u])][x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]) \\
&= \mathcal{T}(C[u][x/u])
\end{aligned}$$

- $u[x/v] \rightarrow_{\text{gc}} u$ .

Let us write  $u$  as  $\lambda\vec{y}.u'$ , where  $u'$  is not a  $\lambda$ -abstraction. As  $\text{fv}(t) = \text{fv}(\mathcal{T}(t))$  and all bound variables are distinct,  $x \notin \text{fv}(u')$ . Then

$$\begin{aligned}
& \mathcal{T}(u[x/v]) \\
&= \text{ALC}(\mathcal{T}(\lambda\vec{y}.u')[x/\mathcal{T}(v)]) \\
&= \text{ALC}((\lambda\vec{y}.\mathcal{T}(u'))[x/\mathcal{T}(v)]) \\
&= \text{ALC}(\lambda\vec{y}.\mathcal{T}(u')[x/\mathcal{T}(v)]) \\
&= \lambda\vec{y}.\text{ALC}(\mathcal{T}(u')[x/\mathcal{T}(v)]) \\
&= \lambda\vec{y}.\mathcal{T}(u')[x/\mathcal{T}(v)] \\
&\rightarrow_{\text{gc}} \lambda\vec{y}.\mathcal{T}(u') \\
&= \mathcal{T}(u)
\end{aligned}$$

$$\begin{aligned} & \mathcal{T}(D[u][x/u]) \\ &= \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)]). \end{aligned}$$

Thus, by Lemma 9.27,

$$\begin{aligned} & \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)]) \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(D[u])). \end{aligned}$$

Thus, by Corollary 9.26,

$$\begin{aligned} & \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])]) \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[u]))]) \end{aligned} \tag{B.1}$$

and

$$\begin{aligned} & \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])] \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[u]))] \end{aligned} \tag{B.2}$$

Let  $\bar{z}$  be fresh for the equations above. Then:

$$\begin{aligned} & \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])]) \\ &= \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])]) \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[u]))][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])]) \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[u]))][\bar{z}/\text{ALC}(\mathcal{T}(D[u]))]) \\ &= \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[u]))][\bar{z}/\text{ALC}(\mathcal{T}(D[u]))]) \end{aligned} \tag{B.3}$$

and

$$\begin{aligned} & \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])] \\ &= \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])] \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[u]))][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])] \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[u]))][\bar{z}/\text{ALC}(\mathcal{T}(D[u]))] \\ &= \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[u]))][\bar{z}/\text{ALC}(\mathcal{T}(D[u]))] \end{aligned} \tag{B.4}$$

**Figure B.22:** Equations for the cases where  $x \notin \text{fv}(D[u])$

$$\begin{aligned} & \mathcal{T}(D[u][x/u]) \\ &= \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)][\bar{x}/\mathcal{T}(u)]). \end{aligned}$$

Thus, by Lemma 9.27,

$$\begin{aligned} & \text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)]) \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)]). \end{aligned}$$

Thus, by Corollary 9.26,

$$\begin{aligned} & \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])]) \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])]) \end{aligned} \quad (\text{B.5})$$

and

$$\begin{aligned} & \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])] \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])] \end{aligned} \quad (\text{B.6})$$

Let  $\bar{z}$  be fresh for the equations above. Then:

$$\begin{aligned} & \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])]) \\ &= \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])]) \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])]) \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])]) \\ &= \text{ALC}(\mathcal{T}(v)[z/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])]) \end{aligned} \quad (\text{B.7})$$

and

$$\begin{aligned} & \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])] \\ &= \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])] \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[x])[x/\mathcal{T}(u)])] \\ & \rightarrow_{\lambda_{\text{ES}}}^+ \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])] \\ &= \text{ALC}(\mathcal{T}(v)[x/\mathcal{T}(u)])[z/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])][\bar{z}/\text{ALC}(\mathcal{T}(D[u])[x/\mathcal{T}(u)])] \end{aligned} \quad (\text{B.8})$$

**Figure B.23:** Equations for the cases where  $x \in \text{fv}(D[u])$

To complete the proof, we induct over the structure of  $t$  to prove the cases of reduction under the root. We treat the more complicated cases.

- $t = u_1[x/u_2] \rightarrow_{\Lambda_{\text{sub}}} u'_1[x/u_2] = t'$ .

By i.h.,  $\mathcal{T}(u_1) \rightarrow_{\lambda_{\text{es}}}^+ \mathcal{T}(u'_1)$ . We break the case over three subcases.

- $x \notin \text{fv}(u_1)$ .

$$\mathcal{T}(u_1[x/u_2]) = \text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)]).$$

$$\text{By Lemma 9.24.1, } \text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)]) \rightarrow_{\lambda_{\text{es}}}^+ \text{ALC}(\mathcal{T}(u'_1)[x/\mathcal{T}(u_2)]) = \mathcal{T}(u'_1[x/u_2]).$$

- $x \in \text{fv}(u_1)$ ,  $x \in \text{fv}(u'_1)$ .

$$\mathcal{T}(u_1[x/u_2]) = \text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)][\bar{x}/\mathcal{T}(u_2)]).$$

By Lemma 9.24.2,  $\text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)]) \rightarrow_{\lambda_{\text{es}}}^+ \text{ALC}(\mathcal{T}(u'_1)[x/\mathcal{T}(u_2)])$  then by Lemma 9.24.1,

$$\begin{aligned} & \text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)][\bar{x}/\mathcal{T}(u_2)]) \\ &= \text{ALC}(\text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)])(\bar{x}/\mathcal{T}(u_2))) \\ &\rightarrow_{\lambda_{\text{es}}}^+ \text{ALC}(\text{ALC}(\mathcal{T}(u'_1)[x/\mathcal{T}(u_2)])(\bar{x}/\mathcal{T}(u_2))) \\ &= \text{ALC}(\mathcal{T}(u'_1)[x/\mathcal{T}(u_2)])(\bar{x}/\mathcal{T}(u_2)) \\ &= \mathcal{T}(u'_1[x/u_2]) \end{aligned}$$

- $x \in \text{fv}(u_1)$ ,  $x \notin \text{fv}(u'_1)$ .

$$\mathcal{T}(u_1[x/u_2]) = \text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)][\bar{x}/\mathcal{T}(u_2)]).$$

By Lemma 9.24.3,  $\text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)]) \rightarrow_{\lambda_{\text{es}}}^+ \mathcal{T}(u'_1)$  then by Lemma 9.24.1,

$$\begin{aligned} & \text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)][\bar{x}/\mathcal{T}(u_2)]) \\ &= \text{ALC}(\text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)])(\bar{x}/\mathcal{T}(u_2))) \\ &\rightarrow_{\lambda_{\text{es}}}^+ \text{ALC}(\mathcal{T}(u'_1)(\bar{x}/\mathcal{T}(u_2))) \\ &=_{\alpha} \mathcal{T}(u'_1[x/u_2]) \end{aligned}$$

- $t = u_1[x/u_2] \rightarrow_{\Lambda_{\text{sub}}} u_1[x/u'_2] = t'$ .

By i.h.,  $\mathcal{T}(u_2) \rightarrow_{\lambda_{\text{es}}}^+ \mathcal{T}(u'_2)$ . We break the proof over two subcases.

- $x \notin \text{fv}(u_1)$ .

$$\mathcal{T}(u_1[x/u_2]) = \text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)]).$$

$$\text{By Lemma 9.26, } \text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)]) \rightarrow_{\lambda_{\text{es}}}^+ \text{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u'_2)]) = \mathcal{T}(u_1[x/u'_2]).$$

–  $x \in \mathbf{fv}(u_1)$ .

$$\mathcal{T}(u_1[x/u_2]) = \mathbf{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)][\bar{x}/\mathcal{T}(u_2)]).$$

By Lemma 9.26,  $\mathbf{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)]) \rightarrow_{\lambda\text{es}}^+ \mathbf{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u'_2)])$ . Let's write  $\mathbf{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)])$  as  $\lambda\vec{y}.v$ , where  $v$  is not a  $\lambda$ -abstraction. Then we have  $\lambda\vec{y}.v \rightarrow_{\lambda\text{es}}^+ \lambda\vec{y}.v' = \mathbf{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u'_2)])$ . Therefore,

$$\begin{aligned} & \mathbf{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u_2)][\bar{x}/\mathcal{T}(u_2)]) \\ &= \lambda\vec{y}.v[\bar{x}/\mathcal{T}(u_2)] \\ &\rightarrow_{\lambda\text{es}}^+ \lambda\vec{y}.v'[\bar{x}/\mathcal{T}(u_2)] \\ &\rightarrow_{\lambda\text{es}}^+ \lambda\vec{y}.v'[\bar{x}/\mathcal{T}(u'_2)] \\ &= \mathbf{ALC}(\mathcal{T}(u_1)[x/\mathcal{T}(u'_2)][\bar{x}/\mathcal{T}(u'_2)]) \\ &= \mathcal{T}(u_1[x/u'_2]). \end{aligned}$$

□

# Appendix C

## Appendix for Part III

*One Minute of Silence* – Soundgarden (arr. John Lennon)

In this appendix, we present an depiction of the Sudoku algorithm in action. We then compare  $\lambda$ BIG and its reduction strategy to previous work in the  $\pi$ -calculus [100] and the fusion calculus [126] and discuss a ‘linear’ representation for  $\lambda$ BIG terms. Next, proofs concerning the safety of  $\sigma\lambda\nu\delta$ -sorting and concerning the models of  $\lambda$ -calculi are presented. We close by considering a perhaps more traditional approach to typing and some thoughts about our interpretation of type preorders as structured sets.

### C.1 Sudoku example

Figures C.1–C.6 depict a worked-through Sudoku puzzle employing the rules of our algorithm.



|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
|   | 7 |   | 3 |   |   |   |   |   |
| 5 |   |   | 4 |   |   |   | 2 |   |
| 8 |   |   | 5 |   | 2 | 9 |   |   |
| 6 | 3 | 5 |   |   |   |   |   | 8 |
|   | 8 |   | 2 | 9 |   |   |   |   |
| 9 | 2 | 7 |   |   |   |   |   | 1 |
| 3 |   |   | 9 |   | 5 | 6 |   |   |
| 2 |   |   | 1 |   |   |   | 3 |   |
|   | 1 |   | 8 |   |   |   |   |   |

Figure C.1: The initial puzzle

|                   |                   |                   |                   |                   |                   |                   |                   |                   |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| 123<br>456<br>789 | 7                 | 123<br>456<br>789 | 3                 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 |
| 5                 | 123<br>456<br>789 | 123<br>456<br>789 | 4                 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 2                 | 123<br>456<br>789 |
| 8                 | 123<br>456<br>789 | 123<br>456<br>789 | 5                 | 123<br>456<br>789 | 2                 | 9                 | 123<br>456<br>789 | 123<br>456<br>789 |
| 6                 | 3                 | 5                 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 8                 |                   |
| 123<br>456<br>789 | 8                 | 123<br>456<br>789 | 2                 | 9                 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 |
| 9                 | 2                 | 7                 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 1                 |                   |
| 3                 | 123<br>456<br>789 | 123<br>456<br>789 | 9                 | 123<br>456<br>789 | 5                 | 6                 | 123<br>456<br>789 | 123<br>456<br>789 |
| 2                 | 123<br>456<br>789 | 123<br>456<br>789 | 1                 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 3                 | 123<br>456<br>789 |
| 123<br>456<br>789 | 1                 | 123<br>456<br>789 | 8                 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 | 123<br>456<br>789 |

Figure C.2: The initial puzzle with marked empty squares

|                   |                   |                           |          |                            |                            |                          |                            |                            |
|-------------------|-------------------|---------------------------|----------|----------------------------|----------------------------|--------------------------|----------------------------|----------------------------|
| <sup>1</sup><br>4 | <b>7</b>          | <sup>12</sup><br>4 6<br>9 | <b>3</b> | <sup>1</sup><br>6<br>8     | <sup>1</sup><br>6<br>8 9   | <sup>1</sup><br>4 5<br>8 | <sup>1</sup><br>4 5 6<br>8 | 4 5 6                      |
| <b>5</b>          | <sup>6</sup><br>9 | <sup>1 3</sup><br>6<br>9  | <b>4</b> | <sup>1</sup><br>6<br>7 8   | <sup>1</sup><br>6<br>7 8 9 | <sup>1 3</sup><br>7 8    | <b>2</b>                   | <sup>3</sup><br>6<br>7     |
| <b>8</b>          | 4 6               | <sup>1 3</sup><br>4 6     | <b>5</b> | <sup>1</sup><br>6<br>7     | <b>2</b>                   | <b>9</b>                 | <sup>1</sup><br>4 6<br>7   | <sup>3</sup><br>4 6<br>7   |
| <b>6</b>          | <b>3</b>          | <b>5</b>                  |          | <sup>1</sup><br>4<br>7     | <sup>1</sup><br>4<br>7     | <sup>2</sup><br>4<br>7   | <sup>4</sup><br>7 9        | <b>8</b>                   |
| <sup>1</sup><br>4 | <b>8</b>          | <sup>1</sup><br>4         | <b>2</b> | <b>9</b>                   | <sup>1 3</sup><br>4 6<br>7 | <sup>3</sup><br>4 5<br>7 | 4 5 6<br>7                 | <sup>3</sup><br>4 5 6<br>7 |
| <b>9</b>          | <b>2</b>          | <b>7</b>                  |          | <sup>3</sup><br>4 5 6<br>8 | <sup>3</sup><br>4 6<br>8   | <sup>3</sup><br>4 5      | 4 5 6                      | <b>1</b>                   |
| <b>3</b>          | 4                 | <sup>4</sup><br>8         | <b>9</b> | <sup>2</sup><br>4<br>7     | <b>5</b>                   | <b>6</b>                 | <sup>1</sup><br>4<br>7 8   | <sup>2</sup><br>4<br>7     |
| <b>2</b>          | 4 5 6<br>9        | <sup>4 6</sup><br>8 9     | <b>1</b> | <sup>4 6</sup><br>7        | <sup>4 6</sup><br>7        | 4 5<br>7 8               | <b>3</b>                   | <sup>4 5</sup><br>7 9      |
| <sup>4</sup><br>7 | <b>1</b>          | <sup>4 6</sup><br>9       | <b>8</b> | <sup>2 3</sup><br>4 6<br>7 | <sup>3</sup><br>4 6<br>7   | <sup>2</sup><br>4 5<br>7 | <sup>4 5</sup><br>7 9      | <sup>2</sup><br>4 5<br>7 9 |

Figure C.3: Remove all marks which are not consistent

|                   |          |                     |          |                          |                          |                          |                            |                            |
|-------------------|----------|---------------------|----------|--------------------------|--------------------------|--------------------------|----------------------------|----------------------------|
| <sup>1</sup><br>4 | <b>7</b> | <sup>12</sup><br>4  | <b>3</b> | <sup>1</sup><br>6<br>8   | <sup>1</sup><br>6<br>8 9 | <sup>1</sup><br>4 5<br>8 | <sup>1</sup><br>4 5 6<br>8 | 4 5 6                      |
| <b>5</b>          | <b>9</b> | <sup>1 3</sup>      | <b>4</b> | <sup>1</sup><br>6<br>7 8 | <sup>1</sup><br>6<br>7 8 | <sup>1 3</sup><br>7 8    | <b>2</b>                   | <sup>3</sup><br>6<br>7     |
| <b>8</b>          | <b>6</b> | <sup>1 3</sup><br>4 | <b>5</b> | <sup>1</sup><br>7        | <b>2</b>                 | <b>9</b>                 | <sup>1</sup><br>4<br>7     | <sup>3</sup><br>4<br>7     |
| <b>6</b>          | <b>3</b> | <b>5</b>            | <b>7</b> | <sup>1</sup><br>4        | <sup>1</sup><br>4        | <sup>2</sup><br>4        | <sup>4</sup><br>9          | <b>8</b>                   |
| <sup>1</sup><br>4 | <b>8</b> | <sup>1</sup><br>4   | <b>2</b> | <b>9</b>                 | <sup>1 3</sup><br>4      | <sup>3</sup><br>4 5<br>7 | 4 5 6<br>7                 | <sup>3</sup><br>4 5 6<br>7 |
| <b>9</b>          | <b>2</b> | <b>7</b>            | <b>6</b> | <sup>3</sup><br>4 5<br>8 | <sup>3</sup><br>4<br>8   | <sup>3</sup><br>4 5      | 4 5                        | <b>1</b>                   |
| <b>3</b>          | <b>4</b> | <sup>8</sup>        | <b>9</b> | <sup>2</sup><br>7        | <b>5</b>                 | <b>6</b>                 | <sup>1</sup><br>7 8        | <sup>2</sup><br>7          |
| <b>2</b>          | <b>5</b> | <sup>6</sup><br>8 9 | <b>1</b> | <sup>4 6</sup><br>7      | <sup>4 6</sup><br>7      | 4<br>7 8                 | <b>3</b>                   | <sup>4</sup><br>7 9        |
| <b>7</b>          | <b>1</b> | <sup>6</sup><br>9   | <b>8</b> | <sup>2 3</sup><br>4 6    | <sup>3</sup><br>4 6      | <sup>2</sup><br>4 5      | <sup>4 5</sup><br>9        | <sup>2</sup><br>4 5<br>9   |

Figure C.4: Finish some squares and repeatedly remove marks

|   |   |                        |   |                        |                          |   |   |                        |
|---|---|------------------------|---|------------------------|--------------------------|---|---|------------------------|
| 1 | 7 | 2                      | 3 | 6                      | 9                        | 5 | 8 | 4                      |
| 5 | 9 | 3                      | 4 | <small>78</small>      | <small>78</small>        | 1 | 2 | 6                      |
| 8 | 6 | 4                      | 5 | 1                      | 2                        | 9 | 7 | 3                      |
| 6 | 3 | 5                      | 7 | <small>4</small>       | <small>1<br/>4</small>   | 2 | 9 | 8                      |
| 4 | 8 | 1                      | 2 | 9                      | <small>3</small>         | 7 | 6 | <small>5</small>       |
| 9 | 2 | 7                      | 6 | 5                      | 8                        | 3 | 4 | 1                      |
| 3 | 4 | <small>8</small>       | 9 | <small>2<br/>7</small> | 5                        | 6 | 1 | <small>2<br/>7</small> |
| 2 | 5 | <small>6<br/>9</small> | 1 | <small>4<br/>7</small> | <small>4 6<br/>7</small> | 8 | 3 | <small>7 9</small>     |
| 7 | 1 | <small>6<br/>9</small> | 8 | <small>23</small>      | <small>3<br/>6</small>   | 4 | 5 | <small>2<br/>9</small> |

Figure C.5: After inferring what some squares must contain

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 1 | 7 | 2 | 3 | 6 | 9 | 5 | 8 | 4 |
| 5 | 9 | 3 | 4 | 8 | 7 | 1 | 2 | 6 |
| 8 | 6 | 4 | 5 | 1 | 2 | 9 | 7 | 3 |
| 6 | 3 | 5 | 7 | 4 | 1 | 2 | 9 | 8 |
| 4 | 8 | 1 | 2 | 9 | 3 | 7 | 6 | 5 |
| 9 | 2 | 7 | 6 | 5 | 8 | 3 | 4 | 1 |
| 3 | 4 | 8 | 9 | 2 | 5 | 6 | 1 | 7 |
| 2 | 5 | 6 | 1 | 7 | 4 | 8 | 3 | 9 |
| 7 | 1 | 9 | 8 | 3 | 6 | 4 | 5 | 2 |

Figure C.6: The solution

## C.2 Proofs for $\Lambda\text{BIG } \sigma\lambda\nu\delta\text{-sorting}$

In this section we assume that the sorting functor has the type  $\mathcal{U}^s : \Lambda\text{BIG}^s \rightarrow \Lambda\text{BIG}^{ks_1}$  for some  $s$ -category  $\Lambda\text{BIG}^{ks_1}$  in the sorting scheme of Figure 11.1. We write **def-node** meaning a node whose control is some typed variant of **def**.

**Lemma C.1.** *If a bigraph  $G$  is  $\sigma\lambda\nu\delta$ -sorted and a root of  $G$  has an exposed **def-node**  $v$  then it has exactly one  $\delta$ -name located at that root. Furthermore, that name is linked to exactly one point under the root, the port of  $v$ .*

*Proof.* All bigraphs in  $\Lambda\text{BIG}^{ks_1}$  have at most one exposed **def-node** per region. If a bigraph has at most one exposed **def-node**, it cannot have an exposed site as this breaks the capacity rule. Therefore,  $G$  has a unique exposed **def-node** with no siblings. By the scoping rule of local bigraphs, the port  $q$  of the **def-node** cannot link to a closed  $\sigma$ -link. Therefore,  $q$  must link to some outer  $\delta$ -name  $x$  at its root. As the **def-node** has no siblings, by L2, P1, and P4 and the kind sorting of  $G$  we conclude that no other  $\delta$ -names can be located at the root. By P5 of  $G$ , we conclude that  $q$  is the only point in  $G$  to link to  $x$ .  $\square$

**Proposition C.2** (composition respects  $\sigma\lambda\nu\delta$ -sorting). *If  $A : H \rightarrow I$  and  $B : I \rightarrow J$  are  $\sigma\lambda\nu\delta$ -sorted and  $B \circ A$  is defined then  $B \circ A$  is sorted.*

*Proof.*

L1: Let  $x_2 : \nu$  be an outer name of  $B \circ A$ . By L1 in  $B$ ,  $x_2$  links to  $\nu$ -ports  $p_1, \dots, p_i$  and  $\nu$ -names  $y_1, \dots, y_l$  in  $B$ . By L1 in  $A$ , all names in  $\{y_1, \dots, y_l\}$  are linked to  $\nu$ -ports  $p_{i+1}, \dots, p_j$  and  $\nu$ -names  $y_{l+1}, \dots, y_m$  in  $A$ .

L2: Let  $x_2 : \delta$  be an outer name of  $B \circ A$ . By L2 in  $B$ ,  $x_2$  links to  $\nu$ -ports  $p_1, \dots, p_i$  and  $\nu$ -names  $y_1, \dots, y_l$  in  $B$ . By L1 in  $A$ , the names in  $\{y_1, \dots, y_l\}$  are linked to  $\nu$ -ports  $p_{i+1}, \dots, p_j$  and  $\nu$ -names  $y_{l+1}, \dots, y_m$  in  $A$ . Composition respects L2 on all these links in  $B \circ A$ . There are two subcases:

- $x_2$  links to a unique  $\delta$ -name  $x_1$  in  $B$ . By L2 in  $A$ ,  $x_1$  links to  $\nu$ -ports  $p_{j+1}, \dots, p_k$  and  $\nu$ -names  $y_{m+1}, \dots, y_n$  in  $A$ . Composition respects L2 on all these links in  $B \circ A$ . By L2 in  $A$  again,  $x_1$  links to exactly one  $\delta$ -point in  $A$  and hence  $x_2$  links to exactly one  $\delta$ -point in  $B \circ A$ .
- $x_2$  links to a unique  $\delta$ -port  $q$  in  $B$ . This satisfies L2 in  $B \circ A$ .

L3: Let  $q$  be a  $\sigma$ -port of  $A$ . L3 is satisfied in  $B \circ A$  since it is satisfied in  $A$ .

Let  $q$  be a  $\sigma$ -port of  $B$ . By L3 in  $B$ ,  $q$  links to  $\nu$ -ports  $p_1, \dots, p_i$  and  $\nu$ -names  $y_1, \dots, y_l$ . By L1 in  $A$ , the names in  $\{y_1, \dots, y_l\}$  are linked to  $\nu$ -ports  $p_{i+1}, \dots, p_j$  and  $\nu$ -names  $y_{l+1}, \dots, y_m$ . Composition respects L3 on all these links in  $B \circ A$ . We have two cases:

- $q$  links to a unique  $\delta$ -name  $x_1$  in  $B$ . By L2 in  $A$ ,  $x_1$  links to  $\nu$ -ports  $p_{j+1}, \dots, p_k$  and  $\nu$ -names  $y_{m+1}, \dots, y_n$  in  $A$ . Composition respects L3 on all these links in  $B \circ A$ . By L2 in  $A$  again,  $x_1$  links to exactly one  $\delta$ -point in  $A$  and hence  $x_2$  links to exactly one  $\delta$ -point in  $B \circ A$ .
- $q$  links to a  $\delta$ -port  $q'$  in  $B$ . This satisfies L3 in  $B \circ A$ .

L4: Let  $q$  be a  $\lambda$ -port of  $A$ . L4 is satisfied in  $B \circ A$  since it is satisfied in  $A$ .

Let  $q$  be a  $\lambda$ -port of  $B$ . By L4 in  $B$ ,  $q$  links to  $\nu$ -ports  $p_1, \dots, p_i$  and  $\nu$ -names  $y_1, \dots, y_l$ . By L1 in  $A$ , the names in  $\{y_1, \dots, y_l\}$  are linked to  $\nu$ -ports  $p_{i+1}, \dots, p_j$  and  $\nu$ -names  $y_{l+1}, \dots, y_m$  in  $A$ . Composition respects L4 on all these links in  $B \circ A$ .

P1: Let  $v_1 <_{B \circ A}^2 v_2$  with  $ctrl(v_1) = \text{def}$  and  $ctrl(v_2) = \text{sub}$ .

If  $v_1 <_B^2 v_2$  or  $v_1 <_A^2 v_2$  then the condition is satisfied. Otherwise, we have  $v_1 <_A r$  and  $r <_B^1 v' <_B^1 v_2$  with  $ctrl(v') = \text{D}$  by the kind sorting. By P3 in  $B$ , there exists an inner name  $x : \sigma$  located at  $r$  which is linked to the port  $p_{v_2}$  of  $v_2$ . Lemma C.1 completes the proof.

P2: Let  $v <_{B \circ A} v_2 <_{B \circ A}^1 v_3$  with  $ctrl(v) = \text{var}$ ,  $ctrl(v_2) = \text{D}$ , and  $ctrl(v_3) = \text{sub}$ .

If the **sub**-node is in  $A$  or all nodes are in  $B$  then the condition is satisfied by P2 in  $A$  and  $B$ . If the **sub**-node and its **def**-node are both in  $B$  then the condition is satisfied by L3, P1, and P6 in  $B$ . Otherwise, we have  $v <_A v_1 <_A^1 r$  and  $r <_B^1 v' <_B^1 v_2$  with  $ctrl(v_1) = \text{def}$  by the kind sorting. By P3 in  $B$ , there exists a unique inner name  $x : \delta$  located at  $r$  which is linked to the port  $p_{v_2}$  of  $v_2$ . Lemma C.1 completes the proof.

P3: Let  $s <_{B \circ A}^1 v_1 <_{B \circ A}^1 v_2$  with  $ctrl(v_1) = \text{D}$  and  $ctrl(v_2) = \text{sub}$ . If  $v_2$  is a node of  $A$  then the condition is satisfied. If  $v_2$  is a node of  $B$  then we must have  $s <_A^1 r$  and  $r <_B^1 v_1 <_B^1 v_2$ . By P3 in  $B$ , there exists an inner name  $x : \sigma$  located at  $r$  which is the only name located at  $r$  to be linked to the port  $p_{v_2}$  of  $v_2$ . By the kind sorting,  $r$  has kind  $(1 \text{ def})^1$ . Therefore,  $r$  contains exactly one site  $s$  in  $A$  and by L2 in  $A$ , there is exactly one  $\delta$ -name  $y$  located at  $s$  which links to  $x$ . Therefore, by P5, no  $\nu$ -name located as  $s$  links to  $x$ . Thus,  $y : \delta$  is the only name located at  $s$  which links to  $p_{v_2}$  in  $B \circ A$ .

P4: Let  $x : \delta$  be an inner name located at a site  $s$  under a **def**-node  $v$  in  $B \circ A$ . If  $v$  is a node of  $A$ , then by P4 in  $A$ ,  $x : \delta$  cannot link to an outer  $\delta$ -name of  $A$  and hence cannot link to an outer  $\delta$ -name in  $B \circ A$ . If  $v$  is a node of  $B$  we must have  $s <_A r <_B v$ . Assume that  $x$  links to an outer  $\delta$ -name in  $B \circ A$ . We must have  $\text{link}_A(x) = y : \delta$ ,  $\text{link}_B(y) = z : \delta$  for some  $y$  and  $z$  with  $y$  located at  $r$ . However, this contradicts P4 in  $B$ .

P5: Let  $z : \delta$  be an outer name of  $B \circ A$ .

Let  $z$  be connected to an inner  $\delta$ -name  $x$  in  $B \circ A$ . We must have  $\text{link}_A(x) = y : \delta$  and  $\text{link}_B(y) = z$ . By P5 in  $A$ ,  $y$  cannot link to any other points which are located in any location of  $x$  in  $A$ . By P5 in  $B$ ,  $z$  cannot link to any other points which are located in any location of  $y$  in  $B$ . Therefore,  $z$  cannot link to any other points which are located in any location of  $x$  in  $B \circ A$ .

Let  $z$  be connected to an  $\delta$ -port  $p$  under root  $t$  in  $B \circ A$ . If  $p$  belongs to a node of  $B$ , then  $z$  does not connect to any other points under  $t$  by P5 in  $B$ . Therefore, it cannot link any inner name located under  $t$  and P5 is satisfied in  $B \circ A$ . If  $p$  belongs to a node of  $A$ , then we must have  $\text{link}(p) = y : \delta$  and  $\text{link}(y) = z$  with  $p$  located under root  $r$  with  $r <_B t$ . By P5 in  $B$ ,  $z$  cannot link to any other points which are located in any location (those under  $t$  in particular) of  $y$  in  $B$ . By P5 in  $A$ ,  $y$  cannot link to any other points under  $r$  besides  $p$ . Therefore,  $z$  cannot link to any other points under  $t$  besides  $p$  in  $B \circ A$ .

P6: Let  $s <_{B \circ A} v_1 <_{B \circ A}^1 v_2$  with  $\text{ctrl}(v_1) = \text{D}$  and  $\text{ctrl}(v_2) = \text{sub}$ . If  $v_2$  is a node of  $A$  then the condition is satisfied. If  $v_2$  is a node of  $B$  then we must have  $s <_A r$  and  $r <_B v_1 <_B^1 v_2$ . By P6 in  $B$ , no inner  $\nu$ -inner names located at  $r$  are bound by the port  $p$  of  $v_2$ . By P3 of  $B$ , there exists a unique  $\delta$ -point located uniquely at  $r$  bound by  $p$ . Therefore, by L2 and P5 in  $A$ , no inner  $\nu$ -inner names located at  $s$  are bound by  $p$  in  $B \circ A$ .

P7: Let  $v$  be a **U**-node of  $B \circ A$ . If  $v$  is a node of  $A$  then P7 holds in  $B \circ A$ . Otherwise  $v$  is a node of  $B$ . Assume a  $\delta$ -name  $x$  is located under  $v$  in  $B \circ A$ . We must have  $s <_A r <_B v$  with  $x$  located in  $s$ . By L1 and L4 of  $A$ ,  $x$  must link either to a  $\sigma$ -port  $p$  or a  $\delta$ -name  $y$  in  $A$ . The former implies P7 of  $B \circ A$ . In the latter case,  $x$  is linked to a  $\sigma$ -port of  $B$  by P7 of  $B$  and hence is linked to a  $\sigma$ -port in  $B \circ A$ .

□

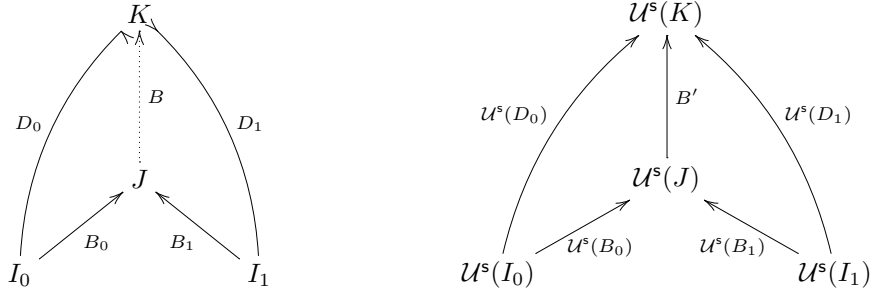
Not all arrows in  $\lambda$ BIG<sup>s</sup> are opcartesian with respect to  $\mathcal{U}^s$ . Consider  $D = \text{id}_{1,x:\delta}$ ,  $A = \text{id}_{1,x:\delta} \oplus y : \nu$  and  $B = (x/x, y)$ .  $B \circ A = D$  with  $A$  and  $D$  sorted but  $B$  is not sorted – P5 is broken. Therefore,

pushout reflection does not immediately follow. Similarly, this implies that  $\mathcal{U}^s$  is not a weak obfibration so that we cannot directly apply the RPO transfer theorem of Birkedal et al. [17]. Instead, we prove that  $\Lambda\text{BIG}^s$  is sufficiently jointly opcartesian with respect to  $\mathcal{U}^s$ .

The problem with the previous example involved idle names. Pushouts in  $\Lambda\text{BIG}^{\text{ks}_1}$  are based on pushouts of local bigraphs and therefore no name of the pushout interface is idle in both arrows of the pushout. We can use this fact and Proposition 5.17 to prove pushout reflection for  $\Lambda\text{BIG}^s$ .

**Lemma C.3** ( $\Lambda\text{BIG}^s$  is sufficiently jointly opcartesian with respect to  $\mathcal{U}^s$ ). *Let  $\vec{B} : \vec{I} \rightarrow J$  be a cospan in  $\Lambda\text{BIG}^s$  such that no name of  $J$  is idle in both  $B_0$  and  $B_1$ . Then  $\vec{B}$  is jointly opcartesian with respect to  $\mathcal{U}^s$ .*

*Proof.* Let  $\vec{D}$  be a cospan in  $\Lambda\text{BIG}^s$  sharing the same domains as  $\vec{B}$  such that a mediator  $B'$  exists from  $\vec{B}^u$  to  $\vec{D}^u$  in  $\Lambda\text{BIG}^{\text{ks}_1}$ . We must prove the existence of a unique mediator  $B$  from  $\vec{B}$  to  $\vec{D}$ . The situation is depicted below.



We define  $B$  as having the place graph and link graph of  $B'$ . We must prove  $B$  is well-sorted. For most of the proof, we need only use properties of  $D_0$  and  $B_0$ .

- L1: Let  $z : \nu$  be an outer name of  $B$ . By L1 in  $D_0$ ,  $z$  cannot link  $\delta$ -ports in  $B$ . Assume  $z$  links a  $\delta$ -name  $y$  in  $B$ . By L2 in  $B_0$ ,  $y$  must link a  $\delta$ -point in  $B_0$  but this contradicts L1 in  $D_0$ .
- L2: Let  $z : \delta$  be an outer name of  $B$ . By L2 in  $D_0$  and  $B_0$ ,  $z$  must link at least one  $\delta$ -point in  $B$ . Assume that  $z$  links a second  $\delta$ -point in  $B$ . Whether this point is a port or a name, L2 in  $B_0$  would contradict L2 in  $D_0$ .
- L3: Let  $p$  be a  $\sigma$ -port of  $B$ . By L3 in  $D_0$  and L1 in  $B_0$ ,  $p$  must link at least one  $\delta$ -point in  $B$ . By L3 in  $D_0$  and L2 in  $B_0$ ,  $p$  cannot link more than one  $\delta$ -point in  $B$ .
- L4: Let  $p$  be a  $\lambda$ -port of  $B$ . By L4 in  $D_0$  and L2 in  $B_0$ ,  $p$  cannot link any  $\delta$ -points.
- P1: This holds by P1 in  $D_0$ .

P2: This holds by P2 in  $D_0$ .

P3: Let  $r <^1 v_1 <^1 v_2$  in  $B$  with  $ctrl(v_1) = \mathbf{D}$  and  $ctrl(v_2) = \mathbf{sub}$ .  $r$  must have sort  $(1\mathbf{def})^1$ . There are two cases.

- $s <_{B_0} r$  and  $s$  has sort  $(1\mathbf{def})^1$ . Therefore,  $s <^1 v_1 <^1 v_2$  in  $D_0$  so by P3 in  $D_0$ , there exists an inner name  $x : \delta$  located at  $s$  which is the only name located at  $s$  to be linked to the port  $p$  of  $v_2$ . Thus by the scoping rule, there must be a name  $y$  located at  $r$  which is linked to  $p$  in  $B$  and  $x$  in  $B_0$ . By L2 in  $B_0$ ,  $y$  has sort  $\delta$  and with P3 of  $D_0$  we can conclude that no other  $\delta$ -name can link to  $p$  in  $B$ .
- $v_0 <_{B_0} r$  with  $ctrl(v_0) = \mathbf{def}$ . By Lemma C.1, there is exactly one  $\delta$ -name  $y$  located at  $r$  and  $y$  is linked to the port  $q$  of  $v_0$ . By P1 in  $D_0$ ,  $y$  must link to  $p$  in  $B$  and so is the only  $\delta$ -name located at  $r$  which is linked to  $p$  in  $B$ .

In both cases, assume  $y' : \nu$  links to  $p$  in  $B$ .  $y' : \nu$  is not idle in both  $B_0$  and  $B_1$ . Therefore, L1 in  $B_0$  or  $B_1$  would contradict P2 or P6 in  $D_0$  or  $D_1$ . Thus,  $y : \delta$  is the only name located at  $r$  to be linked to  $p$  in  $B$ .

P4: Assume an inner name  $y : \delta$  located at  $r$  under a  $\mathbf{def}$ -node  $v$  links to an outer name  $z : \delta$  in  $B$ . By L2 in  $B_0$ ,  $y$  links a  $\mathbf{def}$ -name  $x$  or  $\mathbf{def}$ -port  $p$  in  $B_0$ . The former contradicts P4 in  $D_0$ . If the latter were true then by the kind sorting and P1 in  $B_0$ ,  $p$  must be a port of an exposed  $\mathbf{def}$  node under  $r$ . By the kind sorting, we must have  $r <^2 v' <_B v$  with  $ctrl(v') = \mathbf{sub}$ . However by P1 in  $D_0$ ,  $y : \delta$  must be bound by the port of  $v'$  which is a contradiction.

P5: Let  $z : \delta$  be an outer name of  $B$ .

If  $z$  links a  $\delta$ -port  $p$  in  $B$  then by P1 in  $D_0$  and  $D_1$ ,  $p$  must be a port of an exposed  $\mathbf{def}$  node. By Lemma C.1 in  $D_0$  and  $D_1$  and since no names are idle in both  $B_0$  and  $B_1$ ,  $z$  links no other points in  $B$ .

If  $z$  links an inner  $\delta$ -name  $y$  in  $B$  then by L2 in  $B_0$ ,  $y$  links a  $\delta$ -point  $q$  in  $B_0$ . By L2 in  $D_0$  and  $B_0$ ,  $z$  cannot link any other  $\delta$ -points in  $B$ . By P5 in  $D_0$ ,  $z$  cannot link any  $\nu$ -ports in  $B$ . Assume that  $z$  links a  $\nu$ -name  $y'$  in  $B$ .  $y'$  is not idle in both  $B_0$  and  $B_1$ . Therefore, L1 in  $B_0$  or  $B_1$  would contradict P5 in  $D_0$  or  $D_1$ . Thus, P5 holds in  $B$ .

P6: Let  $s < v_1 <^1 v_2$  with  $ctrl(v_1) = \mathbf{D}$  and  $ctrl(v_2) = \mathbf{sub}$  in  $B$ . Assume that an inner  $\nu$ -name  $y$  located as  $s$  is bound by the port  $p$  of  $v_2$ .  $y$  is not idle in both  $B_0$  and  $B_1$ . Therefore, L1 in  $B_0$  or  $B_1$  would contradict P2 or P6 in  $D_0$  or  $D_1$ . Thus, P6 holds in  $B$ .



P7: Assume there is an inner  $\delta$ -name  $y$  located under a U-node  $v$  in  $B$ . By L2 in  $B_0$ ,  $y$  is linked to a  $\delta$ -name or  $\delta$ -port  $p$  in  $B_0$ . By L1-L4 in  $D_0$ ,  $p$  must be linked to a  $\delta$ -name or  $\delta$ -port in  $D_0$ . The former contradicts P7 in  $D_0$ . The latter, and P1 and P3 of  $D_0$ , implies that P7 holds in  $B$ .

Therefore,  $B$  is a mediator from  $\vec{B}$  to  $\vec{D}$  such that  $\mathcal{U}^s(B) = B'$ . As  $\mathcal{U}^s$  is faithful,  $B$  is unique.

All pushouts in  $\Lambda\text{BIG}^{\text{ks}_1}$  satisfy the property that no name in the pushout interface is idle in both arrows of the pushout. Idleness is reflected by  $\mathcal{U}^s$ . From the above,  $\Lambda\text{BIG}^s$  is sufficiently jointly opcartesian with respect to  $\mathcal{U}^s$ .  $\square$

**Corollary C.4.**  $\mathcal{U}^s$  reflects pushouts.

*Proof.* Follows by Proposition 5.17.  $\square$

A lack of idle names is necessary and sufficient to show an arrow of  $\Lambda\text{BIG}^s$  is opcartesian.

**Corollary C.5.** A bigraph in  $\Lambda\text{BIG}^s$  is opcartesian with respect to  $\mathcal{U}^s$  if and only if it has no idle names.

*Proof.* Let  $A$  be a bigraph in  $\Lambda\text{BIG}^s$  without idle names and  $D$  be a bigraph with the same domain. By Lemma C.3, there exists a unique mediating arrow from the cospan  $(A, A)$  to the cospan  $(D, D)$  when a pure mediator exists from  $(\mathcal{U}^s(A), \mathcal{U}^s(A))$  to  $(\mathcal{U}^s(D), \mathcal{U}^s(D))$  in  $\Lambda\text{BIG}^{\text{ks}_1}$ .

Let  $A : I \rightarrow J$  have an idle name  $x$  located at  $r$ . By L2,  $x$  must be a  $\nu$ -name. We can always construct a bigraph  $B : J \rightarrow K$  such that  $r <_B v_1 <_B^1 v_2$  with  $\text{ctrl}(v_1) = \text{D}$  and  $\text{ctrl}(v_2) = \text{sub}$ ,  $x$  is bound by  $v_2$ , and  $B \circ A$  is sorted (use the identity on the remaining roots and names of  $A$ ). However, P6 is broken in  $B$  by the binding of  $x$ .  $\square$

**Construction C.6** (building a  $\sigma\lambda\nu\delta$ -sorted RPO).

*Proof.* Let  $\vec{A} : H \rightarrow \vec{I}$  have a bound  $\vec{D} : \vec{I} \rightarrow J$  in  $\Lambda\text{BIG}^s$ . We construct an RPO  $(\vec{B}, B)$  for  $\vec{A}$  relative to  $\vec{D}$  as follows.

First, we build an RPO  $(\vec{B}', B')$  for  $\vec{A}^u$  relative to  $\vec{D}^u$  in  $\Lambda\text{BIG}^{\text{ks}_1}$ . We then define  $(\vec{B}, B)$  as having the same link graph and place graph as  $(\vec{B}', B')$ . This amounts to assigning a sort  $\nu$  or  $\delta$  to each name in the shared interface  $I'$  of  $(\vec{B}', B')$ . The assignment is:

$$\begin{aligned} \text{sort}(x) &= \nu && \text{if for all } p \text{ where } \text{link}_{B_i}(p) = x, i \in \{0, 1\}, \text{sort}(p) \neq \delta; \\ \text{sort}(x) &= \delta && \text{otherwise.} \end{aligned}$$

Let  $I$  be the  $\sigma\lambda\nu\delta$ -sorted interface. We first prove that  $\vec{B}$  is sorted. W.l.o.g., we only treat  $B_0$ .

L1: Follows by the assignment of sorts above.

L2: Let  $y$  be a  $\delta$ -name of  $I$ . By the assignment of sorts, there exists a  $\delta$ -point linked to  $y$  in  $B_0$  and/or  $B_1$ .

Assume that  $y$  is idle in  $B_0$ . By the RPO construction of link graphs,  $y$  must be idle in  $A_1$ . Therefore, as  $A_1$  is sorted,  $y$  must have sort  $\nu$  which is a contradiction.

Assume that  $y$  links  $\nu$ -names in  $B_0$  but no  $\delta$ -points. If  $y$  links a  $\delta$ -port  $p_1$  in  $B_1$  then since  $B_0 \circ A_0 = B_1 \circ A_1$ , a  $\nu$ -name linked to  $y$  in  $B_0$  must link  $p_1 : \delta$  in  $A_0$  which contradicts L1 in  $A_0$ . If  $y$  links a  $\delta$ -name  $x_1$  in  $B_1$  then  $x_1$  must link a  $\delta$ -point  $q_1$  in  $A_1$  by L2. If  $q_1$  is a name then since  $B_0 \circ A_0 = B_1 \circ A_1$ , a  $\nu$ -name linked to  $y$  in  $B_0$  must link  $q_1 : \delta$  in  $A_0$  which contradicts L1 in  $A_0$ . If  $q_1$  is a port then  $y$  must link  $q_1 : \delta$  in  $B_0$  which contradicts our assumption.

Assume that  $y$  links a  $\nu$ -port  $p_0$  in  $B_0$  but no  $\delta$ -points. If  $y$  links a  $\delta$ -port  $p_1$  in  $B_1$  then  $y$  must link a name  $x_0$  in  $B_0$  which links  $p_1$  in  $A_0$ . By L1 in  $A_0$ ,  $x_0$  must have sort  $\delta$  which contradicts our assumption. If  $y$  links a  $\delta$ -name  $x_1$  in  $B_1$  then this must link a  $\delta$ -point  $q_1$  in  $A_1$  by L2. We finish similarly to the last case.

Therefore, assume that  $y$  links a  $\delta$ -point in  $B_0$ . If it links more than one  $\delta$ -point then these are linked together in  $D_0$  which contradicts L2 or L3 in  $D_0$ .

Therefore,  $y$  links exactly one  $\delta$ -point in  $B_0$ .

L3: Let  $p$  be a  $\sigma$ -port of  $B_0$ . As  $D_0 = B \circ B_0$ ,  $p$  must link exactly one  $\delta$ -point in  $B_0$ .

L4: Let  $p$  be a  $\lambda$ -port of  $B_0$ . As  $D_0$  is sorted and  $D_0 = B \circ B_0$ ,  $p$  must link no  $\delta$ -points in  $B_0$ .

P1: Let  $v_1 <_{B_0}^2 v_2$  with  $ctrl(v_1) = \text{def}$  and  $ctrl(v_2) = \text{sub}$ . As  $D_0$  is sorted and  $D_0 = B \circ B_0$ , the port of  $v_2$  must bind the port of  $v_1$  in  $B_0$ .

P2: Let  $v_1 <_{B_0} v_2 <_{B_0}^1 v_3$  with  $ctrl(v_1) = \text{var}$ ,  $ctrl(v_2) = \text{D}$ , and  $ctrl(v_3) = \text{sub}$ . As  $D_0$  is sorted and  $D_0 = B \circ B_0$ , the port of  $v_3$  cannot not bind the port of  $v_1$  in  $B_0$ .

P3: Let  $s <_{B_0}^1 v_1 <_{B_0}^1 v_2$  with  $ctrl(v_1) = \text{D}$  and  $ctrl(v_2) = \text{sub}$ . As  $D_0$  is sorted and  $D_0 = B \circ B_0$ , there must exist an inner name  $x : \delta$  located at  $s$  which is the only name located at  $s$  to be linked to the port of  $v_2$  in  $B_0$ .

P4: Assume an inner  $\delta$ -name  $x_0$  located under a  $\text{def}$ -node  $v_1$  links to an outer  $\delta$ -name  $y$  in  $B_0$ . By P4 and L1 in  $D_0$ ,  $link_B(y)$  is not a  $\delta$ -name or  $\nu$ -name. By L4 in  $D_0$ ,  $link_B(y)$  is not a

$\lambda$ -port. Therefore,  $\text{link}_B(y)$  must be a  $\sigma$ -port of some node  $v_2$ . However, as  $y$  is located in a site under  $v_2$  then since  $v_1 <_{D_0} v_2$ : i) if  $v_1$  is located under the left side of  $v_2$  then this contradicts P1 of  $D_0$  as  $D_0$  is kind-sorted ( $v_1 <_{D_0} v_3 <_{D_0} v_2$  with  $v_3$  a sub-node); and ii) if  $v_1$  is located under the right side of  $v_2$  in  $D_0$  then, by the kind sorting, there exists a def-node  $v_2$  (which is possibly  $v_1$ ) whose port links to  $v_2$  by P1 of  $D_0$ . However, as  $x_0$  also links to the port of  $v_2$ , this contradicts L3 in  $D_0$ .

P5: Assume that  $y : \delta$  is an outer name of  $B_0$  which links a  $\delta$ -point  $p_0$  and some other point  $q$  located in a region where  $p_0$  is located. As L2 holds in  $B_0$ ,  $q$  must have sort  $\nu$ . By L1 and L4 in  $D_0$ ,  $\text{link}_B(y)$  must be a  $\delta$ -name or a  $\sigma$ -port of some node  $v$ . The former case contradicts P5 in  $D_0$ . If a location of  $y$  in  $B$  lies under the D node of  $v$  then the latter case contradicts P2 or P6 in  $D_0$ . Otherwise, if a location of  $y$  in  $B$  lies under the U node of  $v$  then if  $p_0$  is a name, P7 is contradicted in  $D_0$ , and if  $p_0$  is a port, then (by the kind sorting) this contradicts P1 in  $D_0$ .

P6: Assume  $s <_{B_0} v_1 <_{B_0}^1 v_2$  with  $\text{ctrl}(v_1) = \text{D}$  and  $\text{ctrl}(v_2) = \text{sub}$  and an inner  $\nu$ -name  $x_0$  located as  $s$  is bound by the port  $p$  of  $v_2$ . This contradicts P6 in  $D_0$ .

P7: Assume an inner  $\delta$ -name is located under a U-node of a sub-node  $v$  in  $B_0$  which is not linked to a  $\sigma$ -port in  $B_0$ . By P7 of  $D_0$ , it must link to a  $\sigma$ -port of a node in  $B$ . However, as  $D_0 = B \circ B_0$ , this node must contain  $v$  which is a contradiction by L3, P1, and P3 of  $D_0$ .

Thus,  $B_0$  and  $B_1$  are sorted. By the RPO construction for link graphs, no name of  $I$  is idle in both  $B_0$  and  $B_1$ . Therefore, by Lemma C.3,  $B$  is also sorted and  $\vec{B}, B$  is a bound for  $\vec{A}$  relative to  $\vec{D}$ .  $\square$

**Proposition C.7** (valid RPO construction). *Construction C.6 builds RPOs in  $'\Lambda\text{BIG}^5$ .*

*Proof.* The construction builds a relative bound by  $\sigma\lambda\nu\delta$ -sorting an RPO of  $'\Lambda\text{BIG}^{\text{ks}1}$ . The proof follows by Lemma C.3 and the faithfulness of  $\mathcal{U}^5$ .  $\square$

### C.3 Comparing $'\Lambda\text{BIG}$ to other encodings of the $\lambda$ -calculus

In this section, we present some of our observations concerning  $'\Lambda\text{BIG}$  and other encodings of the  $\lambda$ -calculus.

### C.3.1 Explicit substitutions in the $\pi$ -calculus

In Milner's  $\pi$ -calculus encoding of the lazy  $\lambda$ -calculus [100], the encoding of an 'environment entry' used to encode a binding of  $x$  to  $t$  is:

$$\llbracket x/t \rrbracket \stackrel{\text{def}}{=} !x(w). \llbracket t \rrbracket w$$

This replicated term keeps the substitution definition alive, within the scope of the substitution, until no more substitutions may be performed. The replicated term may then be garbage-collected through strong bisimilarity. This is similar to the explicit substitution in  $\lambda$ BIG where instantiation keeps the explicit substitution alive and garbage-collection is instead represented with an explicit reaction rule.

### C.3.2 Evaluation strategies

As stated before,  $\lambda$ BIG has full composition of substitutions and active controls. This combination admits the full evaluation strategy of the  $\lambda$ -calculus [8], shown on the top line of Figure C.7. Sub-reduction strategies of this include the lazy strategy, having the rules  $\beta$  and  $\nu$ , the strong lazy strategy, having the rules  $\beta$ ,  $\nu$  and  $\xi$ , the call-by-name strategy having the rules  $\beta$ ,  $\nu$ , and  $\mu$ , and the call-by-value strategy having the rules  $\beta_v$ ,  $\nu_v$ , and  $\mu_v$  where  $V$  denotes a *value*, which is either a variable or an abstraction but not an application.

Milner presented encodings of the call-by-value and lazy reduction strategies in the  $\pi$ -calculus [100]. Parrow and Victor neatly solved the problem of modelling the strong lazy reduction strategy with their introduction of the fusion calculus, a simplification and extension of

$$\begin{array}{cccc}
 \frac{}{(\lambda x.t)u \rightarrow t\{x/u\}} & \frac{t \rightarrow t'}{t u \rightarrow t' u} & \frac{u \rightarrow u'}{t u \rightarrow t u'} & \frac{t \rightarrow t'}{(\lambda x.t) \rightarrow (\lambda x.t')} \\
 (\beta) & (\nu) & (\mu) & (\xi) \\
 \\
 \frac{}{(\lambda x.t)V \rightarrow_v t\{x/u\}} & \frac{t \rightarrow t'}{t u \rightarrow_v t' u} & \frac{u \rightarrow u'}{t u \rightarrow_v t u'} & \\
 (\beta_v) & (\nu_v) & (\mu_v) & 
 \end{array}$$

**Figure C.7:** Full evaluation strategy (top) and the call-by-value evaluation strategy (bottom)

the  $\pi$ -calculus where input and output are symmetrical notions [126]. Their solution is based on Milner's encoding of the lazy reduction strategy.

Bigraphs build on ideas from both calculi so it may not be surprising that it can model the full evaluation strategy (see Theorem 8.8 and Proposition 11.3). Grohmann and Miculan subsequently presented bigraphical models of the the call-by-name and call-by-value strategies based on  $\Lambda\text{BIG}$  using *directed bigraphs* [63].

### C.3.3 Comparing $\lambda\text{lrx}$ and $\Lambda\text{BIG}$

We briefly discuss some parallels between  $\lambda\text{lrx}$  (see Section 7.2.2) and  $\Lambda\text{BIG}$ .

#### Linear $\Lambda\text{BIG}$ terms

A weakening  $W_x(t)$  in  $\lambda\text{lrx}$  adds a free variable  $x$  to a term  $t$  which does not contain an occurrence of  $x$ . We can consider this as adding an idle name to a term  $t$ . This can be modelled in  $\Lambda\text{BIG}$  by the adding of idle names to a term using via extension  $\oplus$ . The property of interface preservation is present in both rewriting systems *i.e.* no free variables are lost during rewriting.

A contraction  $C_x^{y,z}(t)$  can be represented by a substitution bigraph  $x/y \mid x/z$  composed with the  $\Lambda\text{BIG}$  term representing  $t$ . These observations suggest the encoding below from  $\lambda\text{lrx}$  to  $\Lambda\text{BIG}$ , based on Milner's encoding of  $\Lambda_{\text{sub}}$  [107, 111].

$$\begin{aligned}
\llbracket x \rrbracket_{X \uplus x} &\stackrel{\text{def}}{=} \text{var}_x \oplus X \\
\llbracket \lambda x. t \rrbracket_X &\stackrel{\text{def}}{=} (\text{lam}_{(x)} \oplus \text{id}_X)(\llbracket t \rrbracket_{X \uplus x}) \\
\llbracket t \ u \rrbracket_X &\stackrel{\text{def}}{=} (\text{app} \oplus (\text{id}_X \mid \text{id}_X))(\llbracket t \rrbracket_X \mid \llbracket u \rrbracket_X) \\
\llbracket t[x/u] \rrbracket_X &\stackrel{\text{def}}{=} (\text{sub}_{(x)} \oplus \text{id}_X)(\llbracket t \rrbracket_{X \uplus x} \mid (\text{def}_x \oplus \text{id}_X)(\llbracket u \rrbracket_X)) \\
\llbracket W_x(t) \rrbracket_{X \uplus x} &\stackrel{\text{def}}{=} (x \oplus \text{id}_X)(\llbracket t \rrbracket_X) \\
\llbracket C_x^{y,z}(t) \rrbracket_{X \uplus x} &\stackrel{\text{def}}{=} (x/y \mid x/z \oplus \text{id}_X)(\llbracket t \rrbracket_{X \uplus \{y,z\}}).
\end{aligned}$$

Symbolically (*i.e.* syntactically), the encoding generates linear descriptions of  $\Lambda\text{BIG}$  terms where each occurrence of a variable control will be tagged with a different  $\lambda$ -name. For example, the  $\lambda$ -term  $\lambda x. x x$  is translated as a (linear)  $\lambda\text{lrx}$  term as  $\lambda x. C_x^{y,z}(y z)$ . The corresponding bigraph (with index  $\emptyset$ ) is then

$$\text{lam}_{(x)}(x/y \mid x/z)(\text{app} \oplus (\text{id}_{\{y,z\}} \mid \text{id}_{\{y,z\}}))(\text{var}_y \oplus z \mid \text{var}_z \oplus y)$$

which, syntactically, is a linear description. This bigraph may also be written (up to  $\simeq$ ) as

$$\text{lam}_{(x)}(\text{app} \oplus (\text{id}_{\{x\}} \mid \text{id}_{\{x\}}))(\text{var}_x \mid \text{var}_x)$$

which represents the original non-linear term. Examining the two terms above, we may also write

$$\text{lam}_{(x)}(\text{app} \oplus \text{id}_{\{x\}})(\text{id}_1 \mid x/y \mid x/z)(\text{var}_y \oplus z \mid \text{var}_z \oplus y)$$

as an in-between term which has the ‘contraction’ pushed inside as far as possible. This bigraph has no parallel in  $\lambda\text{lr}$  where application is presented as usual with symbolic juxtaposition rather than via an explicit constructor.

Consider the congruence axioms for  $\lambda\text{lr}$  terms and the associated translations with respect to the encoding of  $\lambda\text{lr}$  into  $\lambda\text{BIG}$ . We notice that  $\equiv_A$ ,  $\equiv_{C1_c}$ ,  $\equiv_{C2_c}$ ,  $\equiv_{C_w}$ , and  $\equiv_{Cont_2}$  follow immediately in  $\lambda\text{BIG}$ . Nonetheless, we argue that  $\simeq$ -equivalence classes in the image of the encoding above represent  $\rightarrow_{\text{Ctn}}$  convertible  $\lambda\text{lr}$  terms in  $\rightarrow_{\text{Wk}}$ -normal form.

Finally, Kesner and Lengrand note that weakenings in  $\lambda\text{lr}$  may always be pulled out to the top level allowing efficient garbage collection whereas  $\lambda_{ws}$  cannot pull its labels out to the top-level. In  $\lambda\text{BIG}$ , idle names are always at the top level, substitutions are never propagated through terms, and garbage collection is always at its most efficient.

#### $\lambda\text{BIG}$ with local substitution

The equivalence  $\equiv_S$  which is used for composing substitutions does not follow from the conventional equivalences on bigraphs. This equivalence is used to prove open confluence – confluence on terms with metavariables – for  $\Lambda_{\text{sub}}$ . It does not seem to immediately translate into  $\lambda\text{BIG}$  – encoding  $\equiv_S$  as a pair of reaction rules would trivially lead to infinite reaction sequences. However, it may be worth investigating whether it is possible to: i) define an equivalence relation on the bigraphs in  $\lambda\text{BIG}$  such that a bigraph is related to another if they differ in the order of two adjacent substitutions; and ii) define a bisimulation along this relation.

Let us play devil’s advocate and consider modelling the  $\lambda$ -calculus in bigraphs using the local rules of  $\lambda\text{lr}$  rather than  $\Lambda_{\text{sub}}$ . To that end, we first observe that many reductions disappear through support equivalence. For example, using our translation, the variable  $x$  in the term  $y W_x(t)$  is carried to the outer interface and so  $\llbracket y W_x(t) \rrbracket_{X \uplus \{x, y\}} \simeq \llbracket W_x(y t) \rrbracket_{X \uplus \{x, y\}}$ . This eliminates the need for rules similar to **Weak**<sub>2</sub>, **WLamb**, **WApp**<sub>1</sub>, **WApp**<sub>2</sub>, **WSubs**, and **Cross** in the modified  $\lambda\text{BIG}$ . The rule **Merge** becomes a casualty of composition. Similarly, rules **CLamb**, **CApp**<sub>1</sub>, **CApp**<sub>2</sub>, and **CSubs** are no longer required. We then consider the remaining rules. The rule **Weak**<sub>1</sub> would be similar to  $\rightarrow_D$  – the interface preservation is required by bigraph theory. The **b** rule already has its counterpart in  $\rightarrow_A$ . This leaves the substitution propagation rules **Lamb**, **App**<sub>1</sub>, and **App**<sub>2</sub>, the local copy rule **Var**, the composition rule **Comp**, and finally the duplication rule **Cont**<sub>1</sub>. The modified  $\lambda\text{BIG}$  system would now lie closer to  $\lambda\text{xgc}$  with composition (ignoring the linearity).

## C.4 Proofs for Chapter 11

### C.4.1 Simply typed $\Lambda_{\text{BIG}}$

**Proposition C.8.** *The encoding  $\llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'}$  of a simply typed  $\Lambda_{\text{sub}}$  derivation is well-sorted with outer kind sort  $\mathcal{K}^A$  and link sort  $\Gamma'$ .*

*Proof.* By induction on the derivation of  $t$ . We explain how the derivations of subterms in the inductive cases are valid, then show that the encoding respects the sorting. Note that all  $\Lambda_{\text{BIG}} \rightarrow$ -ions are sorted and that composition preserves kind sorting. We can see from the definition that the outer link sort is  $\Gamma'$  in all cases.

1.  $\llbracket \Gamma, x : A \vdash x : A \rrbracket_{\Gamma'}$ .

All ions are sorted.

2.  $\llbracket \Gamma \vdash t u : A \rrbracket_{\Gamma'}$ .

The last step of the derivation  $\Gamma \vdash t u : A$  must have been an application of **app**. Therefore,  $\Gamma \vdash t : B \rightarrow A$  and  $\Gamma \vdash u : B$  for some  $B$ .

The encodings  $\llbracket \Gamma \vdash t : B \rightarrow A \rrbracket_{\Gamma'}$  and  $\llbracket \Gamma \vdash u : B \rrbracket_{\Gamma'}$  have outer sorts  $\mathcal{K}^{B \rightarrow A}$  and  $\mathcal{K}^B$  respectively and are sorted by the i.h.. Hence, the entire term is sorted.

3.  $\llbracket \Gamma \vdash t[x/u] : A \rrbracket_{\Gamma'}$ .

The last step of the derivation  $\Gamma \vdash t[x/u] : A$  must have been an application of **subs**. Therefore,  $\Gamma, x : B \vdash t : A$  and  $\Gamma \vdash u : B$  for some  $B$ .

The encodings  $\llbracket \Gamma, x : B \vdash t : A \rrbracket_{\Gamma'}$  and  $\llbracket \Gamma \vdash u : B \rrbracket_{\Gamma'}$  have outer sorts  $\mathcal{K}^A$  and  $\mathcal{K}^B$  respectively and are sorted by the i.h.. Hence, the bigraph is kind sorted. Finally, the binding port of the **sub**-node is linked to an inner name  $x$  of the same sort in the composition.

4.  $\llbracket \Gamma \vdash \lambda x. t : A \rightarrow B \rrbracket_{\Gamma'}$ .

The last step of the derivation  $\Gamma \vdash \lambda x. t : A \rightarrow B$  must have been an application of **abs**. Therefore,  $\Gamma, x : A \vdash t : B$ .

The encoding  $\llbracket \Gamma, x : A \vdash t : B \rrbracket_{\Gamma'}$  has outer sort  $\mathcal{K}^B$  and is sorted by the i.h.. Hence, the bigraph is kind sorted. Finally, the binding port of the **lam**-node is linked to an inner name  $x$  of the same sort in the composition.

□

### C.4.2 Intersection typed $\lambda$ BIG

We call ground types and function types *basic types*. We denote multiset sum as  $\sqcup$ , let  $\wedge_n A_i$  denote  $A_1 \wedge \dots \wedge A_n$ , and write  $\wedge S$  for  $\wedge_{A \in S} A$  for some (multi)set  $S$ .

**Definition C.9.** We define two functions as follows:  $P^M$  takes a type to its multiset of basic types,  $P : \tau \rightarrow \mathcal{P}(\tau)$  takes a type to its set of basic types.

$$\begin{aligned} P^M(A) &= \{A\}, \quad A \text{ is basic} & P(A) &= \{A\}, \quad A \text{ is basic} \\ P^M(A \wedge B) &= P^M(A) \uplus P^M(B) & P(A \wedge B) &= P(A) \cup P(B) \end{aligned}$$

Remark that  $P^M(A)$  and  $P(A)$  are non-empty for all types  $A$ .

**Lemma C.10.**  $t \sim \wedge P(t)$ .

*Proof.* By induction over the definition of  $P^M$  and  $P$ , we can see that the underlying set of  $P^M(t)$  is  $P(t)$ . As  $\wedge$  is associative and commutative up to  $\sim$ , we have  $t \sim \wedge P^M(t)$ . By idempotency of  $\wedge$  over  $\sim$  (Lemma 11.45), we have  $t \sim \wedge P^M(t) \sim \wedge P(t)$ .  $\square$

**Lemma C.11.**  $A \ll B$  iff  $P(A) \supseteq P(B)$ .

*Proof.*

$\Rightarrow$  We induct over the derivation of  $A \ll B$ .

1.  $A \ll A$ . Trivial.
2.  $A \wedge B \ll A$ .  $P(A \wedge B) = P(A) \cup P(B) \supseteq P(A)$ .
3.  $A \wedge B \ll B$ . Similar to the previous case.
4.  $A \ll B$  and  $B \ll C$  implies  $A \ll C$ . By the i.h. twice,  $P(A) \supseteq P(B) \supseteq P(C)$ .
5.  $A \ll B$  and  $A \ll C$  implies  $A \ll B \wedge C$ . By the i.h. twice,  $P(A) \supseteq P(B)$  and  $P(A) \supseteq P(C)$ . Therefore,  $P(A) \supseteq P(B) \cup P(C) = P(B \wedge C)$ .

$\Leftarrow$  Let  $P(A) = P(B)$ . Applying Lemma C.10 twice we have:

$$A \sim \wedge P(A) \sim \wedge P(B) \sim B.$$

Let  $P(A) = P(B) + S$  where  $+$  denotes the union of disjoint sets and  $S$  is non-empty. Applying Lemma C.10 twice we have:

$$A \sim \wedge P(A) \sim \wedge(P(B) \uplus S) \sim (\wedge P(B)) \wedge (\wedge S) \ll \wedge P(B) \sim B.$$

In both cases, by the definition of  $\sim$ ,  $A \ll B$ .



□

The following lemma states that the relation  $\ll$  is bounded complete up to  $\sim$  i.e. if two types have a bound then they have a least bound up to  $\sim$ .

**Proposition C.12** ( $\ll_{/\sim}$  is bounded complete). *If  $A_i \ll C$  for all  $i \in \underline{n}$  then there exists a type  $\vee_{\underline{n}} A_i$  such that for all  $D$  where  $A_i \ll D$  for all  $i \in \underline{n}$  we have  $\vee_{\underline{n}} A_i \ll D$ .*

*Proof.* As  $A_i \ll C$ ,  $P(A_i) \supseteq P(C)$  for all  $i$ . Hence,  $\cap_{\underline{n}} P(A_i) \supseteq P(C)$ . As  $P(C)$  is non-empty, we define

$$\vee_{\underline{n}} A_i = \wedge(\cap_{\underline{n}} P(A_i)).$$

Given any type  $D$  with  $A_i \ll D$  for all  $i$  we have  $P(A_i) \supseteq P(D)$ . Hence,  $\cap_{\underline{n}} P(A_i) \supseteq P(D)$ .

Since  $\cap_{\underline{n}} P(A_i)$  consists of ground and function types, we have

$$P(\vee_{\underline{n}} A_i) = P(\wedge(\cap_{\underline{n}} P(A_i))) = \cap_{\underline{n}} P(A_i) \supseteq P(D).$$

Hence, by Lemma C.11,  $\vee_{\underline{n}} A_i \ll D$ . □

**Proposition C.13.** *The encoding  $\llbracket \Gamma \vdash t : A \rrbracket_{\Gamma'}$  of an intersection typed  $\Lambda_{\text{sub}}$  derivation is well-sorted with outer kind sort  $\mathcal{K}^A$  and link sort  $\Gamma'$ .*

*Proof.* By induction on the derivation of  $t$ . We show how the derivations of subterms in the inductive cases are built, then show that the encoding respects the sorting. Note that all  $\Lambda\text{BIG}^\cap$ -ions are sorted and that composition preserves kind sorting. We can see from the definition that the outer link sort is  $\Gamma'$  in all cases.

1.  $\llbracket \Gamma, x : B \vdash x : A \rrbracket_{\Gamma'}$ .

By Lemmas 9.38 and 11.53.1,  $B \ll A$  and  $\mathcal{K}^B \subseteq \mathcal{K}^A$ . Therefore, the inflation is kind sorted and so is the composition.

2.  $\llbracket \Gamma \vdash t u : A \rrbracket_{\Gamma'}$ .

By Lemma 9.38, there exist  $A_i, B_i, i \in \underline{n}$  such that  $\wedge_{\underline{n}}(A_i) \ll A$  and  $\Gamma \vdash t : B_i \rightarrow A_i$  and  $\Gamma \vdash u : B_i$  for  $i \in \underline{n}$ .  $\Gamma \vdash t : \wedge_{\underline{n}}(B_i \rightarrow A_i)$  and  $\Gamma \vdash u : \wedge_{\underline{n}} B_i$  can then be derived using  $\wedge \text{I}$ .

The **app** ion has outer sort  $\mathcal{K}^{\wedge_{\underline{n}} A_i}$ . As  $\wedge_{\underline{n}} A_i \ll A$ ,  $\mathcal{K}^{\wedge_{\underline{n}} A_i} \subseteq \mathcal{K}^A$  by Lemma 11.53.1 and the outer composition is sorted. The encodings  $\llbracket \Gamma \vdash t : \wedge_{\underline{n}}(B_i \rightarrow A_i) \rrbracket_{\Gamma'}$  and  $\llbracket \Gamma \vdash u : \wedge_{\underline{n}} B_i \rrbracket_{\Gamma'}$  have outer sorts  $\mathcal{K}^{\wedge_{\underline{n}}(B_i \rightarrow A_i)}$  and  $\mathcal{K}^{\wedge_{\underline{n}} B_i}$  respectively and are sorted by the i.h.. Hence, the entire term is sorted.

3.  $\llbracket \Gamma \vdash t[x/u] : A \rrbracket_{\Gamma'}$ .

By Lemma 9.38, there exist  $A_i, B_i, i \in \underline{n}$  such that  $\wedge_{\underline{n}} A_i \ll A$  and  $\Gamma, x : B_i \vdash t : A_i$  and  $\Gamma \vdash u : B_i$  for  $i \in \underline{n}$ . We can derive  $\Gamma \vdash u : \wedge_{\underline{n}} B_i$  from the last with applications of  $\wedge$  I. For all  $j$ , we can derive  $\Gamma, x : \wedge_{\underline{n}} B_i \vdash t : A_j$  by adding the derivations  $\Gamma, x : \wedge_{\underline{n}} B_i \vdash x : B_j$ , valid by Corollary 9.33. Thus we can derive  $\Gamma, x : \wedge_{\underline{n}} B_i \vdash t : \wedge_{\underline{n}} A_i$  using  $\wedge$  I.

The **sub** ion has outer sort  $\wedge_{\underline{n}} A_i$ . As  $\wedge_{\underline{n}} A_i \ll A$ ,  $\mathcal{K}^{\wedge_{\underline{n}} A_i} \subseteq \mathcal{K}^A$  by Lemma 11.53.1 and the outer composition is sorted. The encoding  $\llbracket \Gamma, x : \wedge_{\underline{n}} B_i \vdash t : \wedge_{\underline{n}} A_i \rrbracket_{\Gamma', x : \wedge_{\underline{n}} B_i}$  has outer sort  $\mathcal{K}^{\wedge_{\underline{n}} A_i}$  and is sorted by the i.h and the **def** ion has outer sort  $\wedge_{\underline{n}} B_i$  so both may be composed with the **sub** ion. The encoding  $\llbracket \Gamma \vdash u : \wedge_{\underline{n}} B_i \rrbracket_{\Gamma'}$  has outer sort  $\mathcal{K}^{\wedge_{\underline{n}} B_i}$  and is sorted by the i.h.. Hence, the bigraph is kind sorted. Finally, the binding port of the **sub** node is linked to an inner name  $x$  of the same sort in the composition.

4.  $\llbracket \Gamma \vdash \lambda x.t : A \rrbracket_{\Gamma'}$ .

By Lemma 9.38, there exist  $A_i, B_i, i \in \underline{n}$  such that  $\wedge_{\underline{n}} (A_i \rightarrow B_i) \ll A$  and  $\Gamma, x : A_i \vdash t : B_i$  for  $i \in \underline{n}$ . By Corollary 9.33, We can derive  $\Gamma, x : \wedge_{\underline{n}} A_i \vdash t : B_j$  for all  $j \in \underline{n}$  by using the derivations  $\Gamma, x : \wedge_{\underline{n}} A_i \vdash x : A_j$ . Hence, we can derive  $\Gamma, x : \wedge_{\underline{n}} A_i \vdash t : \wedge_{\underline{n}} B_i$  with applications of  $\wedge$  I.

The **lam** ion has outer sort  $\wedge_{\underline{n}} (A_i \rightarrow B_i)$ . As  $\wedge_{\underline{n}} (A_i \rightarrow B_i) \ll A$ ,  $\mathcal{K}^{\wedge_{\underline{n}} (A_i \rightarrow B_i)} \subseteq \mathcal{K}^A$  by Lemma 11.53.1 and the outer composition is sorted. The encoding  $\llbracket \Gamma, x : \wedge_{\underline{n}} A_i \vdash t : \wedge_{\underline{n}} B_i \rrbracket_{\Gamma', x : \wedge_{\underline{n}} A_i}$  has outer sort  $\mathcal{K}^{\wedge_{\underline{n}} B_i}$  and is sorted by the i.h.. Hence, the bigraph is kind sorted. Finally, the binding port of the **lam** node is linked to an inner name  $x$  of the same sort in the composition.

5.  $\llbracket \Gamma \vdash \lambda x.t : B \rightarrow C \rrbracket_{\Gamma'}$ .

By Lemma 9.38,  $\Gamma, x : B \vdash t : C$ . The encoding  $\llbracket \Gamma, x : B \vdash t : C \rrbracket_{\Gamma', x : B}$  has outer sort  $\mathcal{K}^C$  and is sorted by the i.h.. Hence, the bigraph is kind sorted. Finally, the binding port of the **lam** node is linked to an inner name  $x$  of the same sort in the composition.

□

## C.5 Typing $\lambda$ BIG

Our approach in modelling simply typed  $\Lambda_{\text{sub}}$  was to use sortings of bigraphs. Here, we present another idea. We extend interfaces with types  $A \in \tau$  and use the rules of Figure 11.11 – based on those of Figure 11.8 – to identify prime bigraphs which are related to simply typed  $\Lambda_{\text{sub}}$  terms.

Using the rules, we have something approaching a static correspondence. If  $X \supseteq \text{FV}(t)$  and  $\Gamma \vdash t : A$  then we can build a corresponding  $\lambda$ BIG term of type  $A$  and, repeatedly applying the first rule, with a set of idle names  $X \setminus \text{FV}(t)$ .

For dynamic correspondence we require that if  $\Gamma \vdash t : A$  and  $t \rightarrow_{\text{bcgc}} t'$  then if  $G$  is an encoding of  $t$  with type  $A$ ,  $G \rightarrow G'$  with  $G'$  an encoding of  $t'$  with type  $A$ . As reduction can lose variables, the first rule in the table allows these to be added as idle names.

The rules below cannot derive parametric redexes or reactums, only prime bigraphs. However, the reaction relation is defined on ground rules. Rules to allow  $\lambda$ -contexts could be similarly defined.

$$\begin{array}{c}
 \frac{G : I \rightarrow \langle 1, X, A \rangle}{(\text{id} \oplus \{x : A\}) \circ G : I \rightarrow \langle 1, X \uplus \{x : A\}, A \rangle} \qquad \frac{}{\text{var}_{x:A} : \epsilon \rightarrow \langle 1, \{x : A\}, A \rangle} \\
 \\
 \frac{G : I \rightarrow \langle 1, X \uplus \{x : A\}, B \rangle}{(\text{lam}_{(x:A)} \oplus \text{id}_X) \circ G : I \rightarrow \langle 1, X, A \rightarrow B \rangle} \\
 \\
 \frac{F : I \rightarrow \langle 1, X, A \rightarrow B \rangle \quad G : J \rightarrow \langle 1, Y, A \rangle}{(\text{app} \oplus (X|Y))(F \parallel G) : I \parallel J \rightarrow \langle 1, X \cup Y, B \rangle} \\
 \\
 \frac{F : I \rightarrow \langle 1, X \uplus \{x : A\}, B \rangle \quad G : J \rightarrow \langle 1, Y, A \rangle}{(\text{sub}_{(x:A)} \oplus \text{id}_X)(F | (\text{def}_{x:A} \oplus \text{id}_X)G) : I | J \rightarrow \langle 1, X \cup Y, B \rangle}
 \end{array}$$

**Figure C.8:** Typing rules for  $\lambda$ BIG

### C.5.1 Thoughts on the typed models of the $\lambda$ -calculus

In this section we present some (perhaps dubious) intuitions on the models.

The simple type system is a subset of the intersection type system; both the set of types and the rule set of the former are subsets of the latter. Types in the simple type system have no relation to each other whereas intersection types are related via a preorder encoded in the rules (and described with the  $\ll$  relation). We can therefore choose to associate simple types with a set structure and intersection types with a more cohesive set-based structure, say a directed graph.

In earlier work, we described two ‘flavours’ of fitting s-category, partitioned s-categories and meet s-categories. Taking the above set/cohesive structure argument, these are related in a similar

way if we consider the partial order  $(Int, \subseteq)$  where  $Int$  is the set of interface sorts. In the simply typed model, we used a partitioned s-category *i.e.* the intersection of any two distinct interface sorts was empty. The partial order  $(Int, \subseteq)$  in this case is equivalent to a set – no two distinct elements are comparable. In the model with intersection types, the structure forms a bounded complete meet-semilattice.

So just as a simple type system is more discrete than an intersection type system in that there is no relation between types in the former, the same type of relationship holds between interface sorts in their kind sorted encodings.