The Security Picalculus and Non-interference [⋆]

Matthew Hennessy

COGS, University of Sussex, UK

Abstract

The security π-calculus is a typed version of the asynchronous π-calculus in which the types, in addition to constraining the input/output behaviour of processes, have *security levels* associated with them. This enables us to introduce a range of typing disciplines which allow input or output behaviour, or both, to be bounded above or below by a given security level.

We define typed versions of *may* and *must* equivalences for the security π-calculus, where the tests are parameterised relative to a security level. We provide alternative characterisations of these equivalences in terms of *actions in context*; these describe the actions a process may perform in a given typing environment, assuming the observer is constrained by a related, but possibly different, environment.

The paper also contains non-interference results with respect to *may* and *must* testing. These show that certain form of non-interference can be enforced using our typing systems.

Key words: Distributed Systems, Picalculus, security types, non-interference, testing equivalences.

1 Introduction

The asynchronous π-calculus, [3,14], is a simple formalism for describing distributed processes. It presupposes a set of channel names through which processes communicate. Thus $\mathsf{a}?(X)\,P$ is a process which inputs some value v on the channel a, and executes the body P in which X has been substituted by the value v, while output on the same channel is denoted by $a!\langle v\rangle$. These two primitives, together with operators for parallelism, $|$, repetition, $*$, and

[⋆] Research funded by EPSRC grant GR/L93056

channel scoping, $(\mathsf{new}\, n)$, make the π-calculus a very powerful language. For example the term P,

$$*\,\mathsf{req}?(x,y)\ \ (\mathsf{new}\, r)\ \ \mathsf{s}!\langle x,r\rangle \mid r?(z)\, y!\langle z\rangle$$

describes a process which repeatedly receives a request on the channel req, consisting of a value, bound to x, and a return channel, bound to y. This value is in turn sent along the channel s, presumably serviced by some independent server, together with a private return channel r, generated specifically for this purpose. A response is awaited from the service, on the reply channel r, which is then forwarded on the original return channel y.

Numerous typing systems have been developed for this language, [19,24,25]. Most are based on judgements of the form

$$\Gamma \vdash P$$

indicating that the process P is well-typed with respect to the channel environment Γ, which associates capabilities with the free channel names of P. Usually these capabilities are some elaboration of

read capabilities $\mathsf{r}\langle \mathrm{T}\rangle$: the ability to read values of type T from a channel
write capabilities $\mathsf{w}\langle \mathrm{T}\rangle$: the ability to write values of type T to a channel

For example let A denote the tuple type $(\mathbf{int}, \mathsf{w}\langle\mathbf{int}\rangle)$; a value of this type will consist of a pair, the first element of which is an integer, and the second a channel on which integers may be written. If Γ associates the type $\mathsf{r}\langle \mathrm{A}\rangle$ with the channel req and the type $\mathsf{w}\langle \mathrm{A}\rangle$ with s, we would expect the above term, P, to be well-typed with respect to Γ. However for this to be true the local channel r needs to be generated with the write capability $\mathsf{w}\langle\mathbf{int}\rangle$, to be sent along the channel s, and the read capability $\mathsf{r}\langle\mathbf{int}\rangle$, which is used by the process itself. Thus if we were to annotate all bound names and variables with their required types we would obtain the annotated term

$$*\,\mathsf{req}?(x,y):\mathrm{A}\ \ (\mathsf{new}\, r:\mathrm{R})\ \ \mathsf{s}!\langle x,r\rangle \mid r?(z)\, y!(z) \qquad (*)$$

where R is the type $\{\mathsf{w}\langle\mathbf{int}\rangle, \mathsf{r}\langle\mathbf{int}\rangle\}$. This term is well-typed with respect to the above mentioned environment Γ.

Intuitively the use of types constrain the behaviour of processes, ensuring no misuse of channels. By defining sophisticated forms of types process behaviour can be more or less constrained, while at the same time the advantages of well-typing can be preserved. For example a form of polymorphism is investigated in [19], while in [12] security levels are associated with capabilities, to obtain so-called *security types*. Suppose we have two security levels, high, denoted by top, and low, denoted by bot. Then we would have capabilities of the form

$\mathsf{r}_{\mathsf{top}}\langle \mathrm{T}\rangle$, $\mathsf{r}_{\mathsf{bot}}\langle \mathrm{T}\rangle$, $\mathsf{w}_{\mathsf{top}}\langle \mathrm{T}\rangle$, $\mathsf{w}_{\mathsf{bot}}\langle \mathrm{T}\rangle$, where T in turn a security type. By varying the precise definition of a security type we can either implement *resource access control* methodologies, or ensure forms of *non-interference*, [2,9,7]. In this paper we will be concerned with the latter, using a mild variation of the I-types of [12]; essentially types are sets of read/write capabilities, where in addition each capability is annotated by a security level taken from some complete lattice $\langle SL, \preceq, \sqcap, \sqcup, \mathsf{top}, \mathsf{bot}\rangle$. We will refer to the asynchronous π-calculus, augmented with these types, as the *security π-calculus.*

The statement of non-interference results requires some definition of *process behaviour*; intuitively a system is interference-free if it's low level behaviour is independent of changes to high-level behaviour. The main topic of this paper is an investigation of the notion of behaviour of process, relative to a security level, for the security π-calculus.

Process behaviour is relative to some typing environment Γ and therefore we wish to develop a relation of the form

$$\Gamma \triangleright^{\sigma} P \simeq Q$$

meaning, intuitively, that in the typing environment Γ, both P and Q exhibit the same σ-level behaviour. By this we mean that a σ-level observer will be unable to discern a difference between P and Q. For example low-level observers will be unable to see any high-level actions performed by P, Q. But more importantly we assume that these observers are constrained by the typing environment Γ and therefore actions disallowed by this environment will also be invisible to observers.

For example suppose the channel a is not in the domain of Γ. Then we would expect

$$\Gamma \triangleright^{\sigma} \mathsf{a}!\langle v\rangle \mid \mathsf{b}!\langle w\rangle \simeq \mathsf{b}!\langle w\rangle$$

regardless of the value of σ because no observer, well-typed with respect to Γ, will be able to interact with P along the channel a. More generally this will also be true if Γ associates with a only an output capability. Similarly if Γ only associates with it an input capability we will have

$$\Gamma \triangleright^{\sigma} Q \mid \mathsf{a}?(x)\, T \;\simeq\; Q$$

for any process Q.

In this paper we investigate *may* and *must* testing equivalences, [17,10] for the security π-calculus. In particular we give an alternative characterisation of these behavioural equivalences which, as might be expected from [17,10], are based on the sequences of actions that a process can perform. But here these sequences are relative to both a security level and a typing environment.

Unfortunately the situation is even more complicated, as the typing environment of the observer and that of the process being observed may not in general be the same. For example consider the term P, given in $(*)$ above. To be well-typed relative to an environment Γ, Γ needs to associate appropriate types with the free names of P, namely req and s. Now consider a computation involving an observer, also well-typed with respect to Γ, interacting with P. After an interaction on the channel req the process evolves to P_1:

$$(\mathsf{new}\, r : \mathrm{R})\ \mathsf{s}!\langle v, r\rangle \mid r?(z)\, \mathsf{b}!(z)\,,$$

for some value v and channel b sent by the observer. At this stage both the observer and the observed process P_1 can still be typed relative to Γ, as both v and b must have been known to the observer, and therefore be typeable in Γ. However now the observed process generates a new channel r, with type $\mathrm{R} = \{\mathsf{r}\langle\mathbf{int}\rangle, \mathsf{w}\langle\mathbf{int}\rangle\}$. But because of the type associated with s in Γ, r is only sent to the observer with the subtype consisting of the one capability $\mathsf{w}\langle\mathbf{int}\rangle$. Subsequently the observer is working relative to $\Gamma, \mathsf{r}\!:\!\{\mathsf{w}\langle\mathbf{int}\rangle\}$, the environment Γ augmented with a new entry for r, whereas the observed process is working with respect to the *different* environment $\Gamma, \mathsf{r}\!:\!\{\mathsf{r}\langle\mathbf{int}\rangle, \mathsf{w}\langle\mathbf{int}\rangle\}$.
In general the observed process and the observing process will be constrained by related but different environments.

Our characterisation of the behavioural equivalences will be based on what we call a *Context Labelled Transition System.* Here actions take the form

$$\Gamma; \Delta \triangleright P \xrightarrow{\mu}_{\sigma} \Gamma'; \Delta' \triangleright P'$$

indicating that in the typing environment Δ the process P can perform the action μ to interact with some σ-level observer which in turn is type-able in the environment Γ; this action may change the typing environments of both the observer and the observed processes, to Γ' and Δ' respectively. If the type environments Γ and Δ satisfy some minor conditions, (are *compatible*), we say that the above judgement is an *action in context. May* equivalence will be characterised in terms of appropriate sequences of such actions in context while *must* equivalence will also require the development of appropriate notions of *acceptance sets.*

The remainder of the paper is organised as follows. In Section 2 we formally define the syntax of the security π-calculus, together with its (standard) operational semantics. This is followed, in Sub-section 2.3, with a range of typing systems. In the most straightforward we have the judgements

$$\Gamma \vdash P$$

where Γ is a type environment, associating types to channel names and variables. This means that relative to Γ, P uses its channels correctly as input/output devices, ignoring their security annotations. We also have judgements of the form

$$\Gamma \vdash_{\sigma} P$$

which indicates that in addition P uses channels with security level *at most* σ. Similarly we have a typing relation

$$\Gamma \models^{\sigma} P$$

indicating that P uses channels with *at least* security level σ. Indeed we can go further, designing relations such as $\Gamma \vdash_{r\sigma} P$ or $\Gamma \models^{w\sigma} P$ where the *read capabilities* or the *write capabilities* of processes are independently constrained. For all of these typing relations Subject Reduction is easily established.

Section 3 is the heart of the paper. First the behavioural preorders and equivalences are defined, by adapting the standard framework, [17,10], to the security π-calculus. We obtain the relations

$$\Gamma \triangleright_{\sigma} P \simeq_{may} Q$$

and

$$\Gamma \triangleright_{\sigma} P \simeq_{must} Q$$

indicating that P and Q can not be distinguished, relative to *may/must* experiments respectively, by any testing process T such that $\Gamma \vdash_{\sigma} T$, that is any test running at security level *at most* σ, relative to the type environment. This is followed by an exposition of the Context LTS, actions in context and their properties. Sub-section 3.3 then contains an alternative characterisation of $\simeq_{may}$ in terms of sequences of actions in context, while in Subsection 3.4 we give a much more complicated characterisation of $\simeq_{must}$.

One benefit of having behavioural equivalences relativised to security levels is that non-interference results can be stated succinctly. Section 4 contains two such statements, and their proofs. The first gives conditions ensure that

$$\Gamma \triangleright_{\sigma} P \simeq_{may} Q \text{ implies } \Gamma \triangleright_{\sigma} P \mid H \simeq_{may} Q \mid K.$$

It turns out to be sufficient to require that the *read capabilities* of P and Q be bounded above by σ, that is $\Gamma \vdash_{r\sigma} P, Q$, and that the *write capabilities* of H and K be bounded below by some $\delta \not\preceq \sigma$, that is $\Gamma \models^{w\delta} H$, K.

This is quite a general non-interference result. For example in the case where Q is P and K is the empty process $\mathbf{0}$ we obtain

$$\Gamma \triangleright_{\sigma} P \simeq_{may} P \mid H$$

indicating that, under the conditions of the theorem, the process H can not interfere with the behaviour of P.

This result is not true for the *must* equivalence. As explained in Section 4, this is because our types allow contention between processes running at different security levels over read access to channels. However by restricting the type system, allowing only *single level* types, we show that the same result holds for $\simeq_{must}$; these types only allow a channel to be used either for communication between security levels, or for communication within a given security level, but not both.

The paper ends with a brief survey of related work.

2 The Language

In this section we define the language, its operational semantics and the typing system with which we will be concerned.

2.1 The Types

We presuppose a complete lattice $\langle SL, \preceq, \sqcap, \sqcup, \mathsf{top}, \mathsf{bot}\rangle$ of security annotations, ranged over by $\sigma, \rho, \ldots$. For each σ we assume a set of *basic types* at that level, of the form $\mathbf{B}_\sigma$. If the security annotation is omitted, as in **int**, then we assume it has security level bot; as we shall see values of these types are available to all processes. Also, as explained in the Introduction, a σ-level channel type, for channels accessible to processes with security clearance at level σ, consists of a set of σ-level capabilities, i.e. a subset of Cap_σ. These may either be a read capability, of the form $\mathsf{r}_\rho\langle \mathrm{T}\rangle$, for some appropriate ρ and T, or a write capability, of the form $\mathsf{w}_\sigma\langle \mathrm{T}\rangle$. These capabilities are constrained by *consistency requirements*. For example since values with the capability $\mathsf{w}_\sigma\langle \mathrm{T}\rangle$ are written to by σ-level processes we require that T in turn be a σ-level type.

Types, i.e. sets of capabilities, are also constrained. For simplicity in a given type we only allow at most one write capability, and for each level σ at most one read capability at that level. More importantly we ensure that, relative to security levels, only *write-ups*, [9,2], are allowed by requiring that if $\mathsf{w}_\rho\langle \mathrm{T}\rangle$ and $\mathsf{r}_{\rho'}\langle \mathrm{S}\rangle$ are in a type then $\rho \preceq \rho'$; the additional constraint that T be a sub-type of S is well-known [19,13]. The formal definition is as follows:

Definition 2.1. *(Types, Capabilities and Subtyping)* Let $Type_\sigma$, Cap_σ be the least sets, and <:, *consistent* the least relations, which satisfy:

(RT-BASE)

$$\frac{}{\mathbf{B}_\rho \in \mathit{Type}_\sigma}\ \rho \preceq \sigma$$

(RT-WR)

$$\frac{\mathrm{A} \in \mathit{Type}_\sigma}{\mathsf{w}_\sigma\langle \mathrm{A}\rangle \in \mathit{Cap}_\sigma}$$

(RT-RD)

$$\frac{\mathrm{A} \in \mathit{Type}_\rho}{\mathsf{r}_\rho\langle \mathrm{A}\rangle \in \mathit{Cap}_\sigma}\ \sigma \preceq \rho$$

(RT-WRRD)

$$\frac{S \subseteq_{fin} \mathit{Cap}_\sigma}{S \in \mathit{Type}_\sigma}\ S \text{ consistent}$$

(RT-TUP)

$$\frac{\mathrm{A}_i \in \mathit{Type}_\sigma \quad (\forall i)}{(\mathrm{A}_1, \ldots, \mathrm{A}_k) \in \mathit{Type}_\sigma}$$

(U-WR)	$\mathsf{w}_\sigma\langle \mathrm{A}\rangle <: \mathsf{w}_\sigma\langle \mathrm{B}\rangle$	if $\mathrm{B} <: \mathrm{A}$
(U-RD)	$\mathsf{r}_\sigma\langle \mathrm{A}\rangle <: \mathsf{r}_\sigma\langle \mathrm{B}\rangle$	if $\mathrm{A} <: \mathrm{B}$
(U-BASE)	$\mathbf{B}_\sigma <: \mathbf{B}_\rho$	if $\sigma \preceq \rho$
(U-RES)	$\{\mathit{cap}_i\}_{i\in I} <: \{\mathit{cap}'_j\}_{j\in J}$	if $(\forall j)(\exists i)\ \mathit{cap}_i <: \mathit{cap}'_j$
(U-TUP)	$(\mathrm{A}_1, \ldots, \mathrm{A}_k) <: (\mathrm{B}_1, \ldots, \mathrm{B}_k)$	if $(\forall i)\ \mathrm{A}_i <: \mathrm{B}_i$

The set of capabilities Cap is *consistent* if

(a) $\mathsf{w}_\sigma\langle \mathrm{A}\rangle,\ \mathsf{w}_\rho\langle \mathrm{B}\rangle \in \mathrm{Cap}$ implies $\sigma = \rho$ and A is B
(b) $\mathsf{r}_\sigma\langle \mathrm{A}\rangle,\ \mathsf{r}_\sigma\langle \mathrm{B}\rangle \in \mathrm{Cap}$ implies A is B
(c) $\mathsf{w}_\sigma\langle \mathrm{A}\rangle,\ \mathsf{r}_\rho\langle \mathrm{B}\rangle \in \mathrm{Cap}$ implies $\mathrm{A} <: \mathrm{B}$.

These types correspond very closely to the *I-types* of [12]; the rule (RT-RD) ensures that only *write-ups* are allowed, from low-level processes to high-level processes. But we allow multiple read capabilities, which will enable us to be more flexible with respect to allowing/disallowing reading from a channel at different security levels. However subtyping is more restrictive; unlike [12] they can only be sub-typed at the same security level; $\mathsf{r}_\sigma\langle \mathrm{A}\rangle <: \mathsf{r}_\rho\langle \mathrm{B}\rangle$ only if $\sigma = \rho$. Nevertheless this is compensated for in the existence of multiple read capabilities.

Example 2.2.

- The set $\{\mathsf{w}_{\mathsf{bot}}\langle \mathbf{int}\rangle, \mathsf{r}_{\mathsf{bot}}\langle \mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle \mathbf{int}\rangle\}$ is a bot-level channel type, that is an element of $\mathit{Type}_{\mathsf{bot}}$; that is channels of this type may be transmitted on bot-level channels. In turn these channels may be written to by a bot-level process or read by either a bot-level or a top-level process.
- The type $\{\mathsf{w}_{\mathsf{bot}}\langle \mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle \mathbf{int}\rangle\}$ restricts reading from the channel to top-level processes, although bot-level ones can write to it.

- The set $\{\mathsf{w}_{\mathsf{top}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{bot}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle\mathbf{int}\rangle\}$ is not a valid type as it contains a read capability at a lower level than its write capability.
- The set $\{\mathsf{w}_{\mathsf{top}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle\mathbf{int}\rangle\}$ is a top-level type but not a bot-level one; that is, it is in $\mathit{Type}_{\mathsf{top}}$ but not in $\mathit{Type}_{\mathsf{bot}}$.

PROPOSITION 2.3. *For every σ, Type_σ is a preorder with respect to $<:$, with both a partial meet operation $\sqcap$ and a partial join $\sqcup$.*

Proof. The (partial) functions $\sqcap$ and $\sqcup$ are defined by structural induction on types. They are determined by the clauses

$$\begin{aligned}
\mathsf{r}_\sigma\langle \mathrm{A}\rangle \sqcap \mathsf{r}_\sigma\langle \mathrm{A}'\rangle &= \mathsf{r}_\sigma\langle \mathrm{A} \sqcap \mathrm{A}'\rangle \\
\mathsf{r}_\sigma\langle \mathrm{A}\rangle \sqcap \mathsf{r}_\rho\langle \mathrm{A}'\rangle &= \{\mathsf{r}_\sigma\langle \mathrm{A}\rangle, \mathsf{r}_\rho\langle \mathrm{A}\rangle\} \\
\mathsf{w}_\sigma\langle \mathrm{A}\rangle \sqcap \mathsf{w}_\sigma\langle \mathrm{A}'\rangle &= \mathsf{w}_\sigma\langle \mathrm{A} \sqcup \mathrm{A}'\rangle \\
\mathsf{r}_\sigma\langle \mathrm{A}\rangle \sqcup \mathsf{r}_\sigma\langle \mathrm{A}'\rangle &= \mathsf{r}_\sigma\langle \mathrm{A} \sqcup \mathrm{A}'\rangle \\
\mathsf{w}_\sigma\langle \mathrm{A}\rangle \sqcup \mathsf{w}_\sigma\langle \mathrm{A}'\rangle &= \mathsf{w}_\sigma\langle \mathrm{A} \sqcap \mathrm{A}'\rangle
\end{aligned}$$

and these definitions are extended homomorphically to tuple types. □

Multiple read capabilities in a type, such as $\{\mathsf{w}_{\mathsf{bot}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{bot}}\langle\mathbf{int}\rangle, \mathsf{r}_{\mathsf{top}}\langle\mathbf{int}\rangle\}$, allows processes at different security levels to read from the same channel. We can eliminate such contention by using a restricted set of types.

DEFINITION 2.4 (SINGLE-LEVEL TYPES). Let *SlType* be the least set of types obtained by changing the second condition in the definition of *consistent* of Definition 2.1 to read:

$\mathsf{r}_\rho\langle \mathrm{A}\rangle, \mathsf{r}_\sigma\langle \mathrm{B}\rangle \in \mathrm{Cap}$ implies $\rho = \sigma$ and A is B.

Note that these types still allow communication from low-level processes to high-level processes. We leave the reader to check that these types, ordered by $<:$ also has both partial meet and join operations.

2.2 Syntax and Operational Semantics

The syntax of the π-*calculus*, given in Figure 1, uses a predefined set of *names*, ranged over by $a, b, \ldots$ and a set of *variables*, ranged over by x, y, z. *Identifiers* are either variables or names. We also assume a set of *basic values*, ranged over by bv, each of which belong to a given basic type.

The binding constructs $(\mathsf{new}\, a : \mathrm{A})\ Q$ and $u?(X : \mathrm{A})\, Q$ introduce the usual notions of free names and variables, $\mathsf{fn}(P)$ and $\mathsf{fv}(P)$, respectively, and associated

Fig. 1 Syntax

P, Q ::=	*Terms*
$u!\langle v\rangle$	Output
$u?(X:\mathrm{A})\,P$	Input
$\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$	Matching
$(\mathsf{new}\, a:\mathrm{A})\ P$	Name creation
$P \mid Q$	Composition
$*P$	Replication
$\mathbf{0}$	Termination

X, Y ::=	*Patterns*
x	Variable
$(X_1, \ldots, X_k)$	Tuple

u, v, w ::=	*Values*
bv	Base Value
a	Name
x	Variable
$(u_1, \ldots, u_k)$	Tuple

notions of substitution and α-equivalence, $\equiv_\alpha$, are defined as usual. Moreover the typing annotations on the binding constructs are omitted whenever they do not play a role, as will most occurrences of the empty process $\mathbf{0}$.

The behaviour of a process is determined by the interactions in which it can engage. To define these, we give a labelled transition semantics (LTS) for the language. The set *Act* of *actions*, is defined as follows:

μ ::=	*Actions*
τ	Internal action
$a?v$	Input of v on a
$(\tilde{c}:\tilde{\mathrm{C}})a!v$	Output of v on a revealing private names $\tilde{c}$ $(\tilde{c} \subseteq \mathsf{fn}(v))$

Visible actions (all except τ) are ranged over by α, β and if α is an output action we use $\mathcal{E}(\alpha)$ to denote the bound names in α, together with their types: $\mathcal{E}((\tilde{c}:\tilde{\mathrm{C}})a!v) = (\tilde{c}:\tilde{\mathrm{C}})$. Further, let $\mathsf{n}(\mu)$ be the set of *names* occurring in μ, whether free or bound. We say that the actions '$a?v$' and '$(\tilde{c}:\tilde{\mathrm{C}})a!v$' are *complementary*, with $\overline{\alpha}$ denoting a complement of α.

The LTS is defined in Figure 2 and for the most part the rules are straightforward; it is based on the standard operational semantics from [16], to which the reader is referred for more motivation. Note that in the communication rule (L-COM) it is assumed that α is an output action; we omit the corresponding symmetric rule, in which Q performs the output. The last part of the rule (L-CTXT) uses a standard structural congruence over terms. This is defined to be the least equivalence generated by the axioms given in Figure 2, which extends $\equiv_\alpha$ and is preserved by the static operators ($|$ and $(\mathsf{new}\, a)$). Note that because of this rule structurally equivalent processes can perform exactly

Fig. 2 Labelled Transition Semantics

(L-OUT)
$$\frac{}{a!\langle v\rangle \xrightarrow{a!v} \mathsf{0}}$$

(L-IN)
$$\frac{}{a?(X)\,P \xrightarrow{a?v} P\{\!|{}^{v}\!/\!_{X}|\!\}}$$

(L-OPEN)
$$\frac{P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} P'}{(\mathsf{new}\,b:\mathrm{B})\ P \xrightarrow{(b:\mathrm{B})(\tilde{c}:\tilde{\mathrm{C}})a!v} P'}\quad \begin{array}{l} b \neq a \\ b \in \mathsf{fn}(v)\end{array}$$

(L-COM)
$$\frac{P \xrightarrow{\alpha} P',\ \ Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} (\mathsf{new}\,\mathcal{E}(\alpha))\ (P' \mid Q')}$$

(L-EQ)
$$\frac{}{\mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} P} \qquad \frac{}{\mathsf{if}\ u = w\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} Q}\ u \neq w$$

(L-CTXT)
$$\frac{P \xrightarrow{\mu} P'}{*P \quad \xrightarrow{\mu} *P \mid P'} \qquad \frac{P \xrightarrow{\mu} P'}{\begin{array}{l} P \mid Q \xrightarrow{\mu} P' \mid Q \\ Q \mid P \xrightarrow{\mu} Q \mid P'\end{array}}\ \mathsf{bn}(\mu) \notin \mathsf{fn}(Q)$$

$$\frac{P \xrightarrow{\mu} P'}{(\mathsf{new}\,a:\mathrm{A})\ P \xrightarrow{\mu} (\mathsf{new}\,a:\mathrm{A})\ P'}\ a \notin \mathsf{n}(\mu)$$

$$\frac{P \equiv Q,\ Q \xrightarrow{\mu} P'}{P \xrightarrow{\mu} P'}$$

The structural congruence axioms:

(S-NEWNEW)	$(\mathsf{new}\,a)(\mathsf{new}\,b)\ P \equiv (\mathsf{new}\,b)(\mathsf{new}\,a)\ P$ if $a \neq b$
(S-NEWPAR)	$P \mid (\mathsf{new}\,a)\ Q \equiv (\mathsf{new}\,a)\ (P \mid Q)$ if $a \notin \mathsf{fn}(P)$
(S-COMM)	$P \mid Q \equiv Q \mid P$
(S-ZERO)	$P \mid \mathsf{0} \equiv P$

the same set of actions.

We end this sub-section with a result which emphasises the asynchrony of message reception:

LEMMA 2.5 (ASYNCHRONOUS ACTIONS). *If* $P \xrightarrow{(\tilde{c}:\tilde{C})a!v} P'$ *then* $P \equiv (\mathsf{new}\,\tilde{c}:\tilde{\mathrm{C}})\ (a!\langle v\rangle \mid P')$.

Proof. By induction on the derivation of $P \xrightarrow{(\tilde{c}:\tilde{C})a!v} P'$. $\square$

Fig. 3 Typing Rules

(T-ID)
$$\frac{\Gamma(u) <: \mathrm{A}}{\Gamma \vdash u : \mathrm{A}}$$

(T-BASE)
$$\frac{bv \in \mathbf{B}_\sigma}{\Gamma \vdash bv : \mathbf{B}_\sigma}$$

(T-TUP)
$$\frac{\Gamma \vdash v_i : A_i \quad (\forall i)}{\Gamma \vdash (v_1, \ldots, v_k) : (\mathrm{A}_1, \ldots, \mathrm{A}_k)}$$

(T-IN)
$$\frac{\Gamma, X : \mathrm{A} \vdash P \qquad \Gamma \vdash u : \mathsf{r}_\sigma\langle \mathrm{A} \rangle}{\Gamma \vdash u?(X : \mathrm{A})\, P}$$

(T-OUT)
$$\frac{\Gamma \vdash u : \mathsf{w}_\sigma\langle \mathrm{A} \rangle \qquad \Gamma \vdash v : \mathrm{A}}{\Gamma \vdash u!\langle v \rangle}$$

(T-EQ)
$$\frac{\Gamma \vdash u : \mathrm{A}, v : \mathrm{B} \qquad \Gamma \vdash Q \qquad \Gamma \sqcap \{u : \mathrm{B}, v : \mathrm{A}\} \vdash P}{\Gamma \vdash \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q}$$

(T-NEW)
$$\frac{\Gamma, a : \mathrm{A} \vdash P}{\Gamma \vdash (\mathsf{new}\, a : \mathrm{A})\ P}$$

(T-STR)
$$\frac{\Gamma \vdash P,\ Q}{\Gamma \vdash P \mid Q,\ *P,\ \mathbf{0}}$$

2.3 The Typing System

A *type environment* is a finite mapping from identifiers (names and variables) to types. We adopt some standard notation. For example, let $\Gamma, u : \mathrm{A}$ denote the obvious extension of Γ; $\Gamma, u : \mathrm{A}$ is only defined if u is not in the domain of Γ. The subtyping relation $<:$ together with the partial operators $\sqcap$ and $\sqcup$ may also be extended to environments. For example $\Gamma <: \Delta$ if for all u in the domain of Δ, $\Gamma(u) <: \Delta(u)$. We will normally abbreviate the simple environment $\{u : \mathrm{A}\}$ to $u : \mathrm{A}$ and moreover use $v : \mathrm{A}$ to denote its obvious generalisation to values; this is only well-defined when the value v has the same structure as the type A.

The first typing system is given in Figure 3, where the judgements take the form

$$\Gamma \vdash P$$

Intuitively this means that the process P uses all channels as input/output devices in accordance with their types, as given in Γ. It is the standard typing system for the π-calculus, [19,13], adapted to our types; note that the security levels on the capabilities do not play any role. The rule (LT-EQ), which uses the partial meet operator on type environments, is explained in detail in [13], where it is argued to be useful for capability-based type systems, such as ours.

We can also design a type inference system which not only ensures that channels are used according to their types but also controls the security levels of the channels used. One such system is given in Figure 4, where the judgements now take the form

$$\Gamma \vdash_\sigma P$$

This indicates that not only is P well-typed as before but in addition it uses channels with security level *at most* σ. (This corresponds to the typing sys-

Fig. 4 Security Typing Rules

(LT-IN)
$$\frac{\Gamma, X : \mathrm{A} \vdash_{\sigma} P \qquad \Gamma \vdash u : \mathsf{r}_{\delta}\langle \mathrm{A} \rangle}{\Gamma \vdash_{\sigma} u?(X : \mathrm{A})\, P}\ \delta \preceq \sigma$$

(LT-OUT)
$$\frac{\Gamma \vdash v : \mathrm{A} \qquad \Gamma \vdash u : \mathsf{w}_{\delta}\langle \mathrm{A} \rangle}{\Gamma \vdash_{\sigma} u!\langle v \rangle}\ \delta \preceq \sigma$$

(LT-EQ)
$$\frac{\Gamma \vdash u : \mathrm{A}, v : \mathrm{B} \qquad \Gamma \vdash_{\sigma} Q \qquad \Gamma \sqcap \{u : \mathrm{B}, v : \mathrm{A}\} \vdash_{\sigma} P}{\Gamma \vdash_{\sigma} \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q}$$

(LT-NEW)
$$\frac{\Gamma, a : \mathrm{A} \vdash_{\sigma} P}{\Gamma \vdash_{\sigma} (\mathsf{new}\, a : \mathrm{A})\ P}$$

(LT-STR)
$$\frac{\Gamma \vdash_{\sigma} P,\ Q}{\Gamma \vdash_{\sigma} P \mid Q,\ *P,\ \mathbf{0}}$$

tem used in [12].) The only difference is in the input/output rules, where the security level of the channels used are checked. For example $\Gamma \vdash_{\sigma} a!\langle v \rangle$ only if in Γ the channel a can be assigned a security level $\delta \preceq \sigma$, in addition to having the appropriate output capability in Γ.

We can also design a typing system

$$\Gamma \vdash^{\sigma} P$$

which which ensures that P uses channels with security level *at least* σ. The only change is to demand in the input/output rules that $\sigma \preceq \delta$:

(HL-IN)
$$\frac{\Gamma, X : \mathrm{A} \vdash^{\sigma} P \qquad \Gamma \vdash u : \mathsf{r}_{\delta}\langle \mathrm{A} \rangle}{\Gamma \vdash^{\sigma} u?(X : \mathrm{A})\, P}\ \sigma \preceq \delta$$

(HL-OUT)
$$\frac{\Gamma \vdash v : \mathrm{A} \qquad \Gamma \vdash u : \mathsf{w}_{delta}\langle \mathrm{A} \rangle}{\Gamma \vdash^{\sigma} u!\langle v \rangle}\ \sigma \preceq \delta$$

We can provide further mix and matches. For example the type system

$$\Gamma \vdash_{r\sigma} P$$

ensures that all channels from which values are *read* have a read capability of *at most* σ; the security level of the output channels is unexamined. This system is obtained by using the rules in the original Figure 3 but with the rule (T-IN) replaced with (LT-IN); the output rule is left unchanged. In a similar manner we can define relations $\Gamma \vdash_{w\sigma} P$, $\Gamma \vdash^{r\sigma} P$ and $\Gamma \vdash^{w\sigma} P$.

THEOREM 2.6 (SUBJECT REDUCTION). *Let $\Vdash$ represent any of the relations, $\vdash$, $\vdash_{\sigma}$, $\vdash_{r\sigma}$, $\vdash^{r\sigma}$, $\vdash_{w\sigma}$, $\vdash^{w\sigma}$ and suppose $\Delta \Vdash P$. Then*

- $P \xrightarrow{\tau} Q$ *implies* $\Delta \Vdash Q$
- $P \xrightarrow{a?v} Q$ *implies there exists a type* A *such that* $\mathsf{r}_{\delta}\langle \mathrm{A} \rangle \in \Delta(a)$ *and if* $\Delta \sqcap v : \mathrm{A}$ *is well-defined then* $\Delta \sqcap v : \mathrm{A} \Vdash Q$.
  Moreover $\delta \preceq \sigma$ *when* $\Vdash$ *is* $\vdash_{\sigma}$ *or* $\vdash_{r\sigma}$ *and* $\sigma \preceq \delta$ *if it is* $\vdash^{\sigma}$ *or* $\vdash^{r\sigma}$.

- $P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} Q$ *implies there exists a type* A *such that* $\Delta \vdash a : \mathsf{w}_\delta\langle \mathrm{A}\rangle$, $\Delta, \tilde{c} : \tilde{\mathrm{C}} \vdash v : \mathrm{A}$ *and* $\Delta, \tilde{c} : \tilde{\mathrm{C}} \Vdash Q$.
  Moreover $\delta \preceq \sigma$ *when* $\Vdash$ *is* $\vdash_\sigma$ *or* $\vdash_{w\sigma}$ *and* $\sigma \preceq \delta$ *if it is* $\vdash^\sigma$ *or* $\vdash^{w\sigma}$.

Proof. Similar to that of Theorem 3.5 of [12], although in the case of the action $a?v$, the conclusion is a little stronger. However the proof is straightforward. For example consider the case when P is the term $a?(X : \mathrm{B})\, R$, the move is $a?(X)\, R \xrightarrow{a?v} R\{\!|v/x|\!\}$ and $\Delta \vdash_\sigma P$. From the typing rules we have $\Delta \vdash a : \mathsf{r}_\delta\langle \mathrm{B}\rangle$ for some $\delta \preceq \sigma$ and $\Delta, X : \mathrm{B} \vdash_\sigma R$. From the former we know that there exists some $\mathrm{A} <: \mathrm{B}$ such that $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Delta(a)$; from the latter, and Subsumption, we have $\Delta, X : \mathrm{A} \vdash_\sigma R$. A standard Substitution Lemma can now be applied for any v such that $\Delta \sqcap v : \mathrm{A}$ is well-defined to obtain $\Delta \sqcap v : \mathrm{A} \vdash_\sigma R\{\!|v/x|\!\}$.

□

3 Behavioural Theories

In this section we develop two behavioural theories of typed processes, based on the general testing theories of [17,10]. In the first section we adapt the original definitions from [17,10] to our language. This is followed by a subsection defining the Context LTS alluded to in the Introduction. Two further subsections use this LTS to determine the *may* and *must* versions of our behavioural equivalence.

3.1 Testing Processes

A *test* or *observer* is a process with an occurrence of a new reserved resource name ω, used to report success. We let T to range over tests, with the typing rule $\Gamma \vdash_\sigma \omega!\langle\rangle$ for all Γ. When placed in parallel with a process P, a test may interact with P, producing an output on ω if some desired behaviour of P has been observed. We write

$$P \textbf{ may } T$$

$T \mid P \xrightarrow{\tau}^* R$ for some R such that R can *report success*, i.e. $R \xrightarrow{\omega!\langle\rangle}$. The stronger relation

$$P \textbf{ must } T$$

holds when in every computation

$$T \mid P \xrightarrow{\tau} R_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} R_n \xrightarrow{\tau} \ldots$$

there is some $R_k,\ k \geq 0$, which can report success.

We can obtain a testing based behavioural preorder between processes by demanding that they react in a similar manner to a given class of tests. Here

Fig. 5 Context LTS

(c-out)

$$\frac{\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)}{\Gamma;\Delta \triangleright a!\langle v\rangle \xrightarrow{a!v}_\sigma \Gamma \sqcap v:\mathrm{A};\Delta \triangleright \mathbf{0}}\quad \delta \preceq \sigma$$

(c-in)

$$\frac{\Gamma \vdash a:\mathsf{w}_\delta\langle \mathrm{B}\rangle \qquad \Gamma \vdash v:\mathrm{B}}{\Gamma;\Delta \triangleright a?(X:\mathrm{A})\,P \xrightarrow{a?v}_\sigma \Gamma;\Delta \sqcap v:\mathrm{A} \triangleright P\{\!|{}^{v}\!/\!{}_{X}|\!\}}\quad \begin{array}{l}\delta \preceq \sigma\\ \mathrm{B} <: \mathrm{A}\end{array}$$

(c-open)

$$\frac{\Gamma;\Delta,b:\mathrm{B} \triangleright P \xrightarrow{(\tilde{c})a!v}_\sigma \Gamma';\Delta' \triangleright P'}{\Gamma;\Delta \triangleright (\mathsf{new}\,b:\mathrm{B})\;P \xrightarrow{(b)(\tilde{c})a!v}_\sigma \Gamma';\Delta' \triangleright P'}\quad \begin{array}{l}b \neq a\\ b \in \mathsf{fn}(v)\end{array}$$

(c-red)

$$\frac{P \xrightarrow{\tau} P'}{\Gamma;\Delta \triangleright P \xrightarrow{\tau}_\sigma \Gamma;\Delta \triangleright P'}$$

(c-equiv)

$$\frac{\Gamma;\Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma';\Delta' \triangleright P' \qquad P \equiv Q}{\Gamma;\Delta \triangleright Q \xrightarrow{\mu}_\sigma \Gamma;\Delta' \triangleright P'}$$

(c-ctxt)

$$\frac{\Gamma;\Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma';\Delta' \triangleright P'}{\Gamma;\Delta \triangleright *P \xrightarrow{\mu}_\sigma \Gamma';\Delta' \triangleright *P \mid P'}$$

$$\frac{\Gamma;\Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma';\Delta' \triangleright P'}{\begin{array}{l}\Gamma;\Delta \triangleright P \mid Q \xrightarrow{\mu}_\sigma \Gamma';\Delta' \triangleright P' \mid Q\\ \Gamma;\Delta \triangleright Q \mid P \xrightarrow{\mu}_\sigma \Gamma';\Delta' \triangleright Q \mid P'\end{array}}\quad \mathsf{bn}(\mu) \notin \mathsf{fn}(Q)$$

$$\frac{\Gamma,a:\mathrm{A};\Delta,a:\mathrm{A} \triangleright P \xrightarrow{\mu}_\sigma \Gamma',a:\mathrm{A};\Delta',a:\mathrm{A} \triangleright P'}{\Gamma;\Delta \triangleright (\mathsf{new}\,a:\mathrm{A})\;P \xrightarrow{\mu}_\sigma \Gamma;\Delta' \triangleright (\mathsf{new}\,a:\mathrm{A})\;P'}\quad a \notin \mathsf{n}(\mu)$$

we choose the class of tests which are well-typed and use channels from *at most* a given security level σ; that is we require that processes react in the same manner to all tests T such that $\Gamma \vdash_\sigma T$.

Definition 3.1 (Testing Preorders). We write $\Gamma \triangleright_\sigma P \sqsubseteq_{may} Q$ if for every test T such that $\Gamma \vdash_\sigma T$, P **may** T implies Q **may** T.
Similarly $\Gamma \triangleright_\sigma P \sqsubseteq_{must} Q$ means that for every such T, P **must** T implies Q **must** T.
We use $\simeq_{may}$ and $\simeq_{must}$ denote the related equivalence relations.

So for example setting σ to be bot, $\Gamma \triangleright_{\mathsf{bot}} P \simeq_{may} Q$ means that in the type environment Γ, P and Q are indistinguishable by low-level observers, from a may testing point of view.

3.2 The Context Labelled Transition System

It is well-known, [17,10], that testing equivalences are closely related to the ability of processes to perform sequences of actions. We have explained in the Introduction that here we need to relativise these sequences to security levels and to a pair of typing environments, one for the observer and one for the process being observed.

The rules for the Context LTS, are given in Figure 5. The judgements take the form

$$\Gamma; \Delta \triangleright P \xrightarrow{\mu}_{\sigma} \Gamma'; \Delta' \triangleright P'$$

This judgement should be understood as expressing the fact that:

> The process P, in it's current type environment Δ, when run concurrently with any observing process T such that $\Gamma \vdash_{\sigma} T$, can perform the action μ. This will transform P into P' and may also transform the current type environment to Δ' and that of the observing process to Γ'.

Note that judgements are a more explicit form of the *typed actions*, developed in [11] for π-calculus. These actions can take three forms:

internal move: $\Gamma; \Delta \triangleright P \xrightarrow{\tau}_{\sigma} \Gamma; \Delta \triangleright P'$ This corresponds to an internal move by P, which does not depend on its environment. These moves are completely determined by the semantics given in Figure 2; see the rule (C-RED).

input move: $\Gamma; \Delta \triangleright P \xrightarrow{a?v}_{\sigma} \Gamma; \Delta' \triangleright P'$ Here the observing process sends a value v to P along the channel a. The type environment of the observing process does not change, but that of P may be augmented by knowledge of v of which it was previously unaware. An appropriate write capability on a is required of the observing process for the action to take place; see the rule (C-IN).

output move: $\Gamma; \Delta \triangleright P \xrightarrow{(\tilde{c})a!v}_{\sigma} \Gamma'; \Delta' \triangleright P'$ Here P sends a value v along the channel a to the observing process, and typically the observers type environment Γ will be augmented with knowledge of v. However the type environment of P may also be increased by associating with the *new* identifiers $(\tilde{c})$ their declared types; this is implemented in the rule (C-OPEN). Here an appropriate read capability is required of the observing process for the action to take place; see the rule (C-OUT).

The rules in Figure 5 are straightforward and only the first two deserve comment. (C-IN) states that $a?(X : \mathrm{A})\, P$ can receive v along a from a σ-level observer provided the observer has a write capability on a at a level at most σ, and it has the value v at an appropriate type. (C-OUT) is more subtle. In principle the observer could receive v from the observed process $a!\langle v\rangle$ at any type B such that $\Gamma \vdash a : \mathsf{r}_\delta\langle \mathrm{B}\rangle$, where $\delta \preceq \sigma$. However to eliminate much potential nondeterminism in the LTS our rule dictates that for a given $\delta \preceq \sigma$ the observer receives v at the *minimum* B such that $\Gamma \vdash a : \mathsf{r}_\delta\langle \mathrm{B}\rangle$; this is the type A such that $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)$.

Note that in the output actions we do not record the types of the bound names. These we only required in Figure 2 in order to implement communication between processes; see the rule (L-COM). Here we do not need to formalise, at least directly, communication between the process P and its observer.

We can describe precisely the form these judgements can take:

LEMMA 3.2. *Suppose* $\Gamma; \Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma'; \Delta' \triangleright P'$.

$\mu = \tau$*: Here* $\Gamma' = \Gamma$ *and* $\Delta' = \Delta$.
$\mu = a?v$*: Here* $\Gamma' = \Gamma$ *while* $\Delta' = \Delta \sqcap v : \mathrm{A}$ *for some type* A *such that* $\Gamma \vdash v : \mathrm{B},\ a : \mathsf{w}_\delta\langle \mathrm{B}\rangle$*, for some* $\delta \preceq \sigma$ *and* $\mathrm{B} <: \mathrm{A}$
$\mu = (\tilde{c})a!v$*: Here* $\Delta' = \Delta, \tilde{c} : \tilde{C}$ *for some sequence of types* $\tilde{C}$ *such that* $\Delta, \tilde{c} : \tilde{C} \vdash v : \mathrm{A}$*, while* $\Gamma' = \Gamma \sqcap v : \mathrm{A}$ *for some* A *such that* $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)$*, where* $\delta \preceq \sigma$.

Proof. Straightforward rule induction on $\Gamma; \Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma'; \Delta' \triangleright P'$. □

However we are only interested in a subset of the possible judgements which can be derived from the rules in Figure 5. We say that the two type environments Γ and Δ are *compatible* if

- $\Gamma \sqcap \Delta$ exists
- domain(Δ) $\subseteq$ domain(Γ).

The main property of this relation is given by:

LEMMA 3.3. *Suppose* Γ *and* Δ *are compatible. Then* $\Gamma \vdash a : \mathsf{w}_\rho\langle \mathrm{A}\rangle$ *and* $\Delta \vdash a : \mathsf{r}_{\rho'}\langle \mathrm{A}'\rangle$ *imply* $\mathrm{A} <: \mathrm{A}'$ *and* $\rho \preceq \rho'$.

Proof. Simple calculation. □

The triple $\Gamma; \Delta \triangleright P$ is said to be a *configuration* if

- Γ and Δ are compatible
- $\Delta \vdash P$.

When this is the case we will refer to the judgment $\Gamma; \Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma'; \Delta' \triangleright P'$

as an *action in context.*

Configurations are preserved by these actions:

LEMMA 3.4. *If* $\Gamma; \Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma'; \Delta' \triangleright P'$ *is an* action in context *then* $\Gamma'; \Delta' \triangleright P'$ *is a configuration.*

Proof. From Lemma 3.2 we know exactly the form Γ' and Δ' can take, depending on μ. In each case it is straightforward to show that they are compatible. The simplest way to show that $\Delta' \vdash P'$ is using rule induction on $\Gamma; \Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma'; \Delta' \triangleright P'$. □

In future we will limit our attention to judgements $\Gamma; \Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma'; \Delta' \triangleright P'$, which are *actions in context.* This has important consequences, in the case when μ is an output action $(\tilde{c}:\tilde{C})a!v$. It means that the only new names gained by the observer, that is names in the domain of Γ' which are not in that of Γ, are $\tilde{c}$. In other words if w is an identifier in v which does not occur in $\tilde{c}$ the observer already knows about it. However the action may increase the type at which the observer knows w. It is also worth noting that the two rules (C-IN) and (C-OUT) are apriori partial; that is (C-IN) can only be applied if $\Delta \sqcap v:\mathrm{A}$ is well-defined while (C-OUT) requires $\Gamma \sqcap v:\mathrm{A}$ to be well-defined. However it is easy to show that for *actions in context* these environments are in fact well-defined whenever the corresponding premises hold. Moreover in (C-IN) the side-condition $\mathrm{B} <: \mathrm{A}$ may be omitted as it is always satisfied.

We can also determine the circumstances under which the unconstrained actions, from Figure 5, can give rise to *actions in context.*

LEMMA 3.5. *Suppose* $P \xrightarrow{\mu} Q$ *and let* $\Gamma; \Delta \triangleright P$ *be a configuration.*

$\mu = \tau$**:** *Here* $\Gamma; \Delta \triangleright P \xrightarrow{\tau}_\sigma \Gamma; \Delta \triangleright Q$
$\mu = a?v$**:** *Here if* $\Gamma \vdash v:\mathrm{B},\ a:\mathsf{w}_\delta\langle\mathrm{B}\rangle$, *where* $\delta \preceq \sigma$ *then* $\Gamma; \Delta \triangleright P \xrightarrow{a?v}_\sigma \Gamma; \Delta \sqcap v:\mathrm{A} \triangleright Q$ *for some* A *such that* $\mathrm{B} <: \mathrm{A}$.
$\mu = (\tilde{c}:\tilde{C})a!v$**:** *Here if* $\mathsf{r}_\delta\langle\mathrm{A}\rangle \in \Gamma(a)$ *for some* $\delta \preceq \sigma$ *then* $\Gamma; \Delta \triangleright P \xrightarrow{(\tilde{c})a!v}_\sigma \Gamma \sqcap v:\mathrm{A}; \Delta, \tilde{c}:\tilde{\mathrm{C}} \triangleright Q$.

Proof. By rule induction on $P \xrightarrow{\mu} Q$. We examine the case when μ is $(\tilde{c}:\tilde{C})a!v$, where the induction requires a weakening of the hypothesis, namely that $\Delta \vdash P$ and $\Gamma \sqcap \Delta$ exists.

- Suppose $P \xrightarrow{\mu} Q$ is inferred using (L-OUT). We can immediately apply (C-OUT) to obtain the required $\Gamma; \Delta \triangleright P \xrightarrow{a!v}_\sigma \Gamma \sqcap v:\mathrm{A}; \Delta \triangleright Q$, provided $\Gamma \sqcap v:\mathrm{A}$ exists.

  However P has the form $a!\langle v\rangle$ and from $\Delta \vdash P$ we know that $\Delta \vdash$

$v:\mathrm{B}, a:\mathsf{w}_\rho\langle\mathrm{B}\rangle$ for some B. Applying Lemma 3.3 we obtain $\mathrm{B} <: \mathrm{A}$. Then it is easy to show the existence of $\Gamma \sqcap v:\mathrm{A}$ from the fact that Δ and Γ are compatible.

- Suppose $P \xrightarrow{\mu} Q$ is inferred using (L-OPEN), that is

$$(\mathsf{new}\, b:\mathrm{B})\ P' \xrightarrow{(b:\mathrm{B})(\tilde{c}:\tilde{\mathrm{C}})a!v} Q$$

  because $P' \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} Q$.

  $\Delta \vdash P$ implies $\Delta, b:\mathrm{B} \vdash P'$ and the existence of $\Gamma \sqcap \Delta$ also ensures that of $\Gamma \sqcap \Delta, b:\mathrm{B}$. In short the (weaker) inductive hypothesis holds of $\Gamma; \Delta, b:\mathrm{B} \triangleright P'$ and therefore by induction we can obtain the action in context $\Gamma; \Delta, b:\mathrm{B} \triangleright P' \xrightarrow{(\tilde{c})a!v}_\sigma Q$. An application of (C-OPEN) gives the required $\Gamma; \Delta \triangleright P \xrightarrow{(b)(\tilde{c})a!v}_\sigma Q$

$\square$

Note that in *actions in context* $\Gamma; \Delta \triangleright P \xrightarrow{\mu}_\sigma \Gamma'; \Delta' \triangleright Q$ the resulting environments, $\Delta'; \Gamma'$, are not in general determined by Γ and Δ. The change in the environment of the observed process, the change from Δ to Δ', is determined by the declared types of new names introduced by the process. For example consider

$$P_1 = (\mathsf{new}\, c:\mathrm{C}_1)\ a!\langle c\rangle\, \mathbf{0}, \quad P_2 = (\mathsf{new}\, c:\mathrm{C}_2)\ a!\langle c\rangle\, \mathbf{0},$$

where C_i are two different types. Then, assuming Γ, Δ have appropriate capabilities associated with a, we have

$$\Gamma; \Delta \triangleright P_1 \xrightarrow{(c)a!c}_\sigma \Gamma'; \Delta, c:\mathrm{C}_1 \triangleright \mathbf{0}$$
$$\Gamma; \Delta \triangleright P_2 \xrightarrow{(c)a!c}_\sigma \Gamma'; \Delta, c:\mathrm{C}_2 \triangleright \mathbf{0}$$

The reason for this lack of determinism is that the types of bound names are not recorded in the *actions in context*. However were we to record their types we would then have processes which are obviously behaviourally indistinguishable, P_1 and P_2 for example, which would have different *actions in context*.

The lack of determinism of the observers type environment, the change from Γ to Γ', will however play a role in the next section. This arises because of the rule (C-OUT) in Figure 5. In general $\Gamma(a)$ may contain two read capabilities, $\mathsf{r}_{\delta_1}\langle\mathrm{A}_1\rangle$ and $\mathsf{r}_{\delta_2}\langle\mathrm{A}_2\rangle$, in which case Γ' may take either of the forms $\Gamma \sqcap v:\mathrm{A}_1$ or $\Gamma \sqcap v:\mathrm{A}_2$. However by restricting ourselves to single-level types this problem does not arise.

We say Γ is a *single-level* environment if it only uses single-level types. For such environments we can define the partial predicate Γ $after_\sigma$ s by induction on s as follows:

$s = \varepsilon$: Here Γ $after_\sigma$ $s = \Gamma$
$s = a?v \cdot s'$: Here Γ $after_\sigma$ $s = \Gamma$ $after_\sigma$ s'
$s = (\tilde{c})a!v \cdot s'$: Here Γ $after_\sigma$ s is only defined if $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)$ for some $\delta \preceq \sigma$, in which case it is $(\Gamma \sqcap v : \mathrm{A})$ $after_\sigma$ s'.

LEMMA 3.6. *If* $\Gamma; \Delta \triangleright P \xrightarrow{s}_\sigma \Gamma'; \Delta' \triangleright Q$*, where* Γ *is a single-level environment, then* Γ after$_\sigma$ s *is defined and* $\Gamma' = \Gamma$ after$_\sigma$ s.

Proof. By induction on the derivation of $\Gamma; \Delta \triangleright P \xrightarrow{s}_\sigma \Gamma'; \Delta' \triangleright Q$. □

3.3 May testing

In this section we give a characterisation of the relation $\Gamma \triangleright_\sigma P \sqsubseteq_{may} Q$.

Actions in context are generalised to (asynchronous) *traces in context* as follows:

DEFINITION 3.7 (TRACES). Let $\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}_\sigma \Gamma'; \Delta' \triangleright P'$ be the least relation such that:

(TR-τ)
$$\frac{\Gamma; \Delta \triangleright P \xrightarrow{\tau}_\sigma \Gamma'; \Delta' \triangleright P' \qquad \Gamma'; \Delta' \triangleright P' \overset{s}{\Longrightarrow}_\sigma \Gamma''; \Delta'' \triangleright P''}{\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}_\sigma \Gamma''; \Delta'' \triangleright P''}$$

(TR-ϵ)
$$\frac{}{\Gamma; \Delta \triangleright P \overset{\epsilon}{\Longrightarrow}_\sigma \Gamma; \Delta \triangleright P}$$

(TR-α)
$$\frac{\Gamma; \Delta \triangleright P \xrightarrow{\alpha}_\sigma \Gamma'; \Delta' \triangleright P' \qquad \Gamma'; \Delta' \triangleright P' \overset{s}{\Longrightarrow}_\sigma \Gamma''; \Delta'' \triangleright P''}{\Gamma; \Delta \triangleright P \overset{\alpha \cdot s}{\Longrightarrow}_\sigma \Gamma''; \Delta'' \triangleright P''}$$

(TR-ASYNC)
$$\frac{\Gamma \vdash v : \mathrm{A} \qquad \Gamma; \Delta \sqcap v : \mathrm{A} \sqcap a : \mathsf{w}_\delta\langle \mathrm{A}\rangle \triangleright P \mid a!\langle v\rangle \overset{s}{\Longrightarrow}_\sigma \Gamma''; \Delta'' \triangleright P''}{\Gamma; \Delta \triangleright P \overset{a?v \cdot s}{\Longrightarrow}_\sigma \Gamma''; \Delta'' \triangleright P''}\ \delta \preceq \sigma$$

Note that, for simplicity, we have allowed some redundancy here. The rule (TR-α), where α is an input action $a?v$, can actually be derived from (TR-ASYNC) and (TR-τ).

We now show how interactions between a process P and a σ-level observer T, that is a computation from $T \mid P$, can be decomposed into a *trace in context* from P and the complementary sequence from T. It will become clear that it is sufficient to only consider new*free* observers, that is observers which contain no occurrence of the binders $(\mathsf{new}\, a)$.

THEOREM 3.8 (TRACE DECOMPOSITION). *Let* $\Gamma; \Delta \triangleright P$ *be a configuration and suppose* $T \mid P \xrightarrow{\tau}{}^{*} R$ *for some* new*free observer* T *such that* $\Gamma \vdash_{\sigma} T$. *Then there exists a* trace in context

$$\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}_{\sigma} \Gamma'; \Delta' \triangleright P'$$

and a derivation $T \overset{\bar{s}}{\Longrightarrow} T'$*, where* R *has the form* $(\mathsf{new}\, \tilde{c}\!:\!\tilde{C})\ (T' \mid P')$.

Proof. By induction on the derivation of $T|P \xrightarrow{\tau}{}^{*} R$. Consider the non-trivial case when this is of the form $T|P \xrightarrow{\tau}\xrightarrow{\tau}{}^{*} R$. There are essentially three cases:

- Output from T to P. In this case we have $T \xrightarrow{a!v} T_1$, $P \xrightarrow{a?v} P_1$ and $T_1|P_1 \xrightarrow{\tau}{}^{*} R$.

  $\Gamma \vdash_{\sigma} T$ means $\Gamma \vdash v\!:\!\mathrm{B},\ a\!:\!\mathsf{w}_{\delta}\langle \mathrm{B}\rangle$, for some $\delta \preceq \sigma$ and B, and so we may apply Lemma 3.5 to obtain the action in context

  $$\Gamma; \Delta \triangleright P \xrightarrow{a?v}_{\sigma} \Gamma; \Delta \sqcap v:\mathrm{A} \triangleright P_1$$

  for some $\mathrm{B} <: \mathrm{A}$. Moreover the compatibility of Γ and $\Delta \sqcap v:\mathrm{A}$ follows from that of Γ and Δ.

  Subject Reduction implies that $\Gamma \vdash_{\sigma} T_1$ and therefore we may apply induction to obtain

  $$\Gamma; \Delta \triangleright P_1 \overset{s'}{\Longrightarrow}_{\sigma} \Gamma'; \Delta' \triangleright P' \text{ and } T_1 \overset{\bar{s'}}{\Longrightarrow} T'$$

  where R has the form $(\mathsf{new}\, \tilde{c}\!:\!\tilde{C})\ (T' \mid P')$. The required s is $a?v \cdot s'$.
- Output from P to T. In this case we have $T \xrightarrow{a?v} T_1$, $P \xrightarrow{(\tilde{c}:\tilde{C})a!v} P_1$, and $T_1 \mid P_1 \xrightarrow{\tau}{}^{*} R_1$, where R has the form $(\mathsf{new}\, \tilde{c}\!:\!\tilde{C})\ R_1$.

  Here $\Gamma \vdash_{\sigma} T$ implies $\Gamma \vdash \mathsf{r}_{\delta}\langle \mathrm{A}\rangle$ for some $\delta \preceq \sigma$ and so we can apply Lemma 3.5 to obtain the action in context

  $$\Gamma; \Delta \triangleright P \xrightarrow{(\tilde{c})a!v}_{\sigma} \Gamma \sqcap v:\mathrm{A}; \Delta, \tilde{c}\!:\!\tilde{C} \triangleright P_1.$$

  Also by Subject Reduction we know $\Gamma \sqcap v:\mathrm{A} \vdash_{\sigma} T_1$. So we may apply induction to obtain a *trace in context*

  $$\Gamma \sqcap v:\mathrm{A}; \Delta, \tilde{c}\!:\!\tilde{C} \triangleright P_1 \overset{s'}{\Longrightarrow}_{\sigma} \Gamma'; \Delta' \triangleright P',$$

  and the reduction $T_1 \overset{\bar{s'}}{\Longrightarrow} T'$. The required s in this case is $(\tilde{c})a!v \cdot s'$.
- Internal actions by P or T. In this case a simple argument by induction suffices.

□

The converse is more straightforward:

Theorem 3.9 (Trace Composition). *Suppose* $\Gamma;\Delta \triangleright P \overset{s}{\Longrightarrow}_{\sigma} \Gamma';\Delta' \triangleright P'$ *and* $T \overset{\bar{s}}{\Longrightarrow} T'$ *for some* s*. Then there exists a derivation* $T \mid P \xrightarrow{\tau}{}^{*} R$*, where* R *has the form* $(\mathsf{new}\,\tilde{c}:\tilde{C})\ (T' \mid P')$*.*

Proof. By induction on s. □

Refering to the statement of this theorem note that Subject Reduction ensures that $\Delta' \vdash P'$. However in general we do *not* have that $\Gamma' \models_{\sigma} T'$, even under the assumption $\Gamma \models_{\sigma} T$.

Example 3.10. Let P, T be the processes $(\mathsf{new}\,c:\mathrm{C})\ \ a!\langle c\rangle$ and $a?(x:\mathrm{A}_2)\ \ x!\langle\rangle$ respectively and let Γ, Δ map a to the type $\{\mathsf{r}_{\delta_1}\langle \mathrm{A}_1\rangle,\ \mathsf{r}_{\delta_2}\langle \mathrm{A}_2\rangle,\ \mathsf{w}_{\mathsf{bot}}\langle \mathrm{C}\rangle\}$, where A_1, A_2, C are the types $\mathsf{r}_{\mathsf{bot}}\langle\rangle$, $\mathsf{w}_{\mathsf{bot}}\langle\rangle$, $\{\mathrm{A}_1, \mathrm{A}_2\}$ respectively; here we assume $\delta_i \preceq \sigma$. Then

$$\begin{array}{l}
\Gamma \models_{\sigma} T \\
\Delta \vdash P \\
\Gamma;\Delta \triangleright P \xrightarrow{(c)a!c}_{\sigma} \Gamma, c:\mathrm{A}_1;\Delta' \triangleright \mathbf{0} \\
T \xrightarrow{a?c} c!\langle\rangle
\end{array}$$

but $\Gamma, c:\mathrm{A}_1 \not\models_{\sigma} c!\langle\rangle$.

The problem lies, again, with the use of multi-level types, as in the example a may be read at the two levels, δ_1, δ_2.

Lemma 3.11. *Let* Γ *be a single-level environment. Suppose* $\Gamma \models_{\sigma} T$ *and* $\Gamma\,\mathrm{after}_{\sigma}$ s *is defined. Then* $T \overset{\bar{s}}{\Longrightarrow} T'$ *implies* Γ after_{σ} $s \models_{\sigma} T'$*.*

Proof. By induction on s. □

This Lemma may now be applied to the conditions of the Trace Composition Theorem, Theorem 3.9, to ensure when Γ is a single-level environment we can also conclude that $\Gamma' \models_{\sigma} T'$; here Γ' can only be Γ *after*$_{\sigma}$ s.

We may now state a sufficient condition to ensure two processes are related with respect to *may* testing.

Definition 3.12. For any configuration $\mathcal{C}$ let $Aseq^{\sigma}(\mathcal{C}) = \{\, s \mid \mathcal{C} \overset{s}{\Longrightarrow}_{\sigma} \,\}$

PROPOSITION 3.13. *Suppose* $\Delta \vdash P,\ Q$, *where* Γ *and* Δ *are compatible. Then* $\mathrm{Aseq}^\sigma(\Gamma; \Delta \triangleright P) \subseteq \mathrm{Aseq}^\sigma(\Gamma; \Delta \triangleright Q)$ *implies* $\Gamma \triangleright_\sigma P \sqsubseteq_{may} Q$.

Proof. First notice that to show $\Gamma \triangleright_\sigma P \sqsubseteq_{may} Q$ it is sufficient to consider $\mathsf{new}\mathit{free}$ observers T such that $\Gamma \vdash_\sigma T$. For given any other test T' there exists $\mathsf{new}\mathit{free}$ T such that $T' \equiv (\mathsf{new}\,\tilde{c}\!:\!\tilde{C})\ T$ and $T' \mid P$ can eventually report a success if and only if $T \mid P$ can.

So suppose P **may** T, where T is a $\mathsf{new}\mathit{free}$ test such that $\Gamma \vdash_\sigma T$. Then there exists a computation $T|P \xrightarrow{\tau}{}^* R$, where R can report a success. Because $\Gamma; \Delta \triangleright P$ is a configuration Theorem 3.8 can be used to obtain the decomposition into a *trace in context*

$$\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}_\sigma \Gamma'; \Delta' \triangleright P'$$

and a sequence $T \overset{\bar{s}}{\Longrightarrow} T'$, where R has the form $(\mathsf{new}\,\tilde{c}\!:\!\tilde{C})\ (T' \mid P')$.

Since $\mathit{Aseq}^\sigma(\Gamma; \Delta \triangleright P) \subseteq \mathit{Aseq}^\sigma(\Gamma; \Delta \triangleright Q)$ there exists a corresponding *trace in context* from Q,

$$\Gamma; \Delta \triangleright Q \overset{s}{\Longrightarrow}_\sigma \Gamma'; \Delta' \triangleright Q'.$$

Trace Composition, Theorem 3.9, can now be used to recombine this with $T \overset{\bar{s}}{\Longrightarrow} T'$ to obtain a successful computation from $T \mid Q$. $\square$

To prove the converse we need to design tests which can detect the ability of processes to perform *traces in context*. Specifically we will construct a test $T(\Gamma, s, \sigma)$, a $\mathsf{new}\mathit{free}$ process such that $\Gamma \vdash_\sigma T(\Gamma, s, \sigma)$, with the property that P **may** $T(\Gamma, s, \sigma)$ if and only if there is some Δ such that $\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}$. Note Δ will not be used in the definition and the tests will only be defined for certain combinations of Γ and s.

For convenience we only consider traces in which only simple identifiers are output, rather than vectors; that is the output actions are of the form $a!v$ or $(c)a!c$. The generalisation to general output actions of the form $(\tilde{c}\!:\!\tilde{C})a!v$ is very straightforward, but notationally complex. The definition of $T(\Gamma, s, \sigma)$ is by induction on s.

ε: $T(\Gamma, \varepsilon, \sigma)$ is $\omega!\langle\rangle$.

$a?v \cdot s$: In this case the test is defined only if

- there exists some $\delta \preceq \sigma$ such that $\Gamma \vdash a : \mathsf{w}_\delta\langle \mathrm{A}\rangle$ for some type A such that $\Gamma \vdash v : \mathrm{A}$
- $T(\Gamma, s, \sigma)$ is defined.

If this is the case then $T(\Gamma, a?v \cdot s, \sigma)$ is defined to be

$$a!\langle v\rangle \mid T(\Gamma, s, \sigma).$$

$a!v \cdot s$: Here the test is defined if

· $v \in \mathrm{domain}(\Gamma)$
· there exists some type A such that $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)$ for some $\delta \preceq \sigma$
· $T(\Gamma \sqcap v : \mathrm{A}, s, \sigma)$ is defined.
For each such A let $T_{\mathrm{A}}(\Gamma, a!v \cdot s, \sigma)$ be the test

$$a?(x : \mathrm{A})\ \mathsf{if}\ x = v\ \mathsf{then}\ T(\Gamma \sqcap v : \mathrm{A}, s, \sigma)\ \mathsf{else}\ \mathbf{0}$$

Then the required test is

$$T_{\mathrm{A}_1}(\Gamma, a!v \cdot s, \sigma) \oplus \ldots \oplus T_{\mathrm{A}_k}(\Gamma, a!v \cdot s, \sigma)$$

where $\mathrm{A}_1 \ldots \mathrm{A}_k$ is the set of all types which satisfy the conditions above and $\oplus$ represents an *internal choice* operator. This is easily definable by

$$T \oplus U = (\mathsf{new}\, c : \mathrm{C})\ (c!\langle\rangle \mid c?()\, T \mid c?()\, U)$$

where C is the type $\{\mathsf{w}_\sigma\langle\rangle, \mathsf{r}_\sigma\langle\rangle\}$.

$(c)a!c \cdot s$: Here the test is defined if
· there exists some type A such that $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma$ for some $\delta \preceq \sigma$
· $T(\Gamma, c : \mathrm{A}, s, \sigma)$ is defined.
Here again $T(\Gamma, (c)a!c \cdot s, \sigma)$ has the form

$$T_{\mathrm{A}_1}(\Gamma, (c)a!c \cdot s, \sigma) \oplus \ldots T_{\mathrm{A}_k}(\Gamma, (c)a!c \cdot s, \sigma)$$

where A_i range over all the types satisfying these conditions. For such an A, $T_{\mathrm{A}}(\Gamma, (c)a!c \cdot s, \sigma)$ given by

$$a?(x : \mathrm{A})\ \mathsf{if}\ x \in I(\Gamma, \mathrm{A})\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ (T(\Gamma, c : \mathrm{A}, s, \sigma))\{\!|x/c|\!\}$$

where $I(\Gamma, \mathrm{A})$ is the finite set of identifiers $\{\, u \mid \Gamma \vdash u : \mathrm{A}\,\}$ and $\mathsf{if}\ x \in I\ \mathsf{then}\ P\ \mathsf{else}\ Q$ represents the obvious nested if then else structure.

The required properties of these tests are collected in the following Lemmas.

LEMMA 3.14. *If $T(\Gamma, s, \sigma)$ is defined then*

- $\Gamma \vdash_\sigma T(\Gamma, s, \sigma)$
- $T(\Gamma, s, \sigma) \overset{\bar{s}}{\Longrightarrow} R$, *where* $R \overset{w!\langle\rangle}{\longrightarrow}$

Proof. By a straightforward induction on s, although there are considerable details to be checked. For example when s has the form $a!v \cdot s'$ then for $\Gamma \vdash_\sigma T(\Gamma, s, \sigma)$ to be true it is essential that v to be in the domain of Γ. $\square$

LEMMA 3.15. *If there exists some P and some Δ such that $\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}$ then $T(\Gamma, s, \sigma)$ is defined.*

Proof. By induction on the judgement $\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}$. As an example we consider one case, when it has the form

$$\Gamma; \Delta \triangleright P \xrightarrow{a!v} \Gamma \sqcap v : \mathrm{A}; \Delta \triangleright P' \overset{s'}{\Longrightarrow}$$

where $\mathsf{r}_\delta\langle \mathrm{A} \rangle \in \Gamma$ for some A and $\delta \preceq \sigma$.

By induction we know $T(\Gamma \sqcap v : \mathrm{A}, s', \sigma)$ is defined. From Subject Reduction we know $\Delta \vdash v : \mathrm{A}$ and since Γ and Δ are compatible we have that v is in the domain of Γ. So for at least one A the test $T_{\mathrm{A}}(\Gamma, a!v \cdot s', \sigma)$ is defined. It follows that $T(\Gamma, a!v \cdot s', \sigma)$ is also defined. $\square$

It therefore follows from the Composition Theorem that $\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}$ implies P **may** $T(\Gamma, s, \sigma)$. We also have the converse:

Lemma 3.16. *Suppose* $T(\Gamma, s, \sigma)$ *exists and* $\Gamma; \Delta \triangleright P$ *is a configuration. Then* P ***may*** $T(\Gamma, s, \sigma)$ *implies* $\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}$.

Proof. By induction on s, and by way of example we consider the case when it has the form $a!v \cdot s'$.

By examining the form of $T(\Gamma, s, \sigma)$ it must be that $P \xrightarrow{\tau}{}^* \xrightarrow{a!v} P'$ for some P' such that P' **may** $T(\Gamma \sqcap v : \mathrm{A}, s', \sigma)$ for some A such that $\mathsf{r}_\delta\langle \mathrm{A} \rangle \in \Gamma(a)$, where $\delta \preceq \sigma$.

Subject Reduction means $\Delta \vdash P'$ and therefore it is easy to check that $\Gamma \sqcap v : \mathrm{A}; \Delta \triangleright P'$ is a configuration. So we may apply induction to obtain

$$\Gamma \sqcap v : \mathrm{A}; \Delta \triangleright P' \overset{s'}{\Longrightarrow}$$

Lemma 3.5 gives

$$\Gamma; \Delta \triangleright P \xrightarrow{\tau}{}^* \xrightarrow{a!v} \Gamma \sqcap v : \mathrm{A}; \Delta \triangleright P'$$

and the result now follows by (TR-α) in Definition 3.7. $\square$

It therefore follows that

Theorem 3.17 (Alternative Characterisation of May Testing). *Suppose* $\Delta \vdash P, Q$, *and* Γ *is compatible with* Δ. *Then* $\Gamma \triangleright_\sigma P \sqsubseteq_{may} Q$ *if and only if* $\mathrm{Aseq}^\sigma(\Gamma; \Delta \triangleright P) \subseteq \mathrm{Aseq}^\sigma(\Gamma; \Delta \triangleright Q)$.

Proof. The sequence of Lemmas establishes that whenever $\Gamma; \Delta \triangleright P$ is a configuration,

$$\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow} \text{ if and only if } (T(\Gamma, s, \sigma) \text{ exists and } P \textbf{ may } T(\Gamma, s, \sigma)).$$

This is all that is required to prove the converse of Proposition 3.13. $\square$

3.4 Must Testing

In this section examine the relation $\Gamma \triangleright_\sigma P \lesssim_{must} Q$; in particular we give necessary and sufficient criteria for ensuring $\Gamma \triangleright_\sigma P \lesssim_{must} Q$, based on *traces in context*.

The extra ingredients required to capture must testing, in addition to traces, are well-known from [17,10]; they include a *convergence predicate*, indicating that a process has no internal infinite computations, and *acceptance sets*, indicating the next possible actions in which a process can engage. Here these need to be generalised from processes to configurations; they must also be relativised to security levels.

Definition 3.18 (Convergence). We say the configuration $\mathcal{C}$ *converges*, written $\mathcal{C} \Downarrow$, if there is no infinite sequence of derivations

$$\mathcal{C} \xrightarrow{\tau} \mathcal{C}_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} \mathcal{C}_k \xrightarrow{\tau}$$

This relation is then parameterised to *sequences in context* and security levels by

ε: $\mathcal{C} \Downarrow^\sigma$ if $\mathcal{C} \Downarrow$
$s = (\tilde{c})a!v \cdot s'$: $\mathcal{C} \Downarrow^\sigma s$ if $\mathcal{C} \Downarrow$ and whenever $\mathcal{C} \overset{(\tilde{c})a!v}{\Longrightarrow}_\sigma \mathcal{C}'$, $\mathcal{C}' \Downarrow^\sigma s'$.
$s = a?v \cdot s'$: $\mathcal{C} \Downarrow^\sigma s$ if

- $\Gamma \vdash a : \mathsf{w}_\delta\langle \mathrm{A} \rangle, v : \mathrm{A}$ for some $\delta \preceq \sigma$
- $\Gamma; \Delta \sqcap v : \mathrm{A} \sqcap a : \mathsf{w}_\delta\langle \mathrm{A} \rangle \triangleright a!\langle \mathrm{A} \rangle \mid P \ \Downarrow^\sigma s'$

Note that the requirements in the input case are taken directly from the rule (TR-ASYNC). Note also that for a configuration $\Gamma; \Delta \triangleright P$ whether or not it converges is actually independent of the typing environments Γ and Δ; it is only dependent on the semantics of P as given in Figure 2. However convergence relative to a *sequence in context* is in general dependent on these environments.

We now adapt the definition of *Acceptance sets*, [10], to the security π-calculus. First let

$$\mathcal{O}^\sigma(\mathcal{C}) = \{\, a! \mid \exists v. \mathcal{C} \overset{a!v}{\Longrightarrow}_\sigma \,\}$$

and

$$\mathcal{R}^\sigma(\mathcal{C}) = \{\, a? \mid \exists v. \mathcal{C} \overset{a?v}{\Longrightarrow}_\sigma \,\} \cup \mathcal{O}^\sigma(\mathcal{C}).$$

Definition 3.19 (Acceptance sets). For a configuration $\mathcal{C}$, let $A^\sigma(\mathcal{C}, s)$, its σ-level acceptance set after s , be defined by

$$\{\, \mathcal{R}^\sigma(\mathcal{C}') \mid \mathcal{C} \overset{s}{\Longrightarrow}_\sigma \mathcal{C}' \overset{\tau}{\not\longmapsto} \,\}$$

Similarly let its *output* acceptance set after s be given by

$$\{\, \mathcal{O}^\sigma(\mathcal{C}') \mid \mathcal{C} \overset{s}{\Longrightarrow}_\sigma \mathcal{C}' \overset{\tau}{\not\longmapsto} \,\}$$

Note only acceptance sets from *stable* configurations, configurations $\mathcal{C}'$ such that $\mathcal{C}' \overset{\tau}{\not\longmapsto}$, are used.

The security π-calculus is *asynchronous* and therefore, as explained in [4], acceptance sets are too discriminating, when used to characterise *must* testing; to see this it is sufficient to consider the simple example

$$a?(x)\,\mathbf{0} \sqsubseteq_{must} \mathbf{0}\,.$$

The same reference goes on to explain that the use of *output* acceptance sets must also be relativised to sets of input actions, which we now explain.

Input Completions. We use $I^\sigma(\mathcal{C})$ to denote the set of input actions which the configuration $\mathcal{C}$ can perform at level σ, $\{\, a?v \mid \mathcal{C} \overset{a?v}{\longrightarrow}_\sigma \,\}$. More generally we use I to denote an arbitrary *multi-set* of input actions, $c(I)$ to denote $\{\, a? \mid a?v \in I \,\}$ and $\overline{c}(I)$ its converse, $\{\, a! \mid a?v \in I \,\}$.

Then $\searrow^\sigma_I$ is defined to be the least relation which satisfies

- $\mathcal{C} \overset{\tau}{\not\longmapsto}$ and $I^\sigma(\mathcal{C}) \cap I = \emptyset$ implies $\mathcal{C} \searrow^\sigma_I \mathcal{C}$
- $\mathcal{C} \overset{i}{\Longrightarrow}_\sigma \mathcal{C}'$ and $\mathcal{C}' \searrow^\sigma_I \mathcal{C}''$ implies $\mathcal{C} \searrow^\sigma_{I \uplus \{i\}} \mathcal{C}''$.

Intuitively $\mathcal{C}' \searrow^\sigma_I \mathcal{C}'$ means that $\mathcal{C}$ can evolve to a stable configuration $\mathcal{C}'$ by performing a subset of the input actions in the multi-set I; moreover this subset is maximal in the sense that $\mathcal{C}'$ can not perform any of the remaining actions.

Definition 3.20 (Asynchronous acceptance sets). For a configuration $\mathcal{C}$, let $O^\sigma_I(\mathcal{C}, s)$, its σ-level asynchronous acceptance set after s, relative to the multi-set of input actions I, be defined by

$$\{\, \mathcal{O}^\sigma(\mathcal{C}'') \mid \mathcal{C} \overset{s}{\Longrightarrow}_\sigma,\ \mathcal{C}' \searrow^\sigma_I \mathcal{C}'' \,\}.$$

With one final notational convention we can mimic the alternative characterisation of must testing from [4]. We write Γ *allows*$_\sigma$ $a?v$ if $\Gamma \vdash_\sigma a!\langle v\rangle$; this is generalised to sets of actions in the normal manner.

Definition 3.21. Let $\mathcal{C}$, $\mathcal{D}$ be configurations of the form $\Gamma; \Delta \triangleright P$, $\Gamma; \Delta' \triangleright Q$ respectively. Then $\mathcal{C} \ll^{\sigma} \mathcal{D}$ if for every s,

$$\begin{aligned}
\mathcal{C} \Downarrow s \text{ implies } & a)\ \mathcal{D} \Downarrow s \\
& b)\ \forall D \in \mathcal{A}^{\sigma}(\mathcal{D}, s), \forall I \text{ such that } c(I) \cap D = \emptyset \text{ and} \\
& \qquad\qquad (\Gamma\ \mathit{after}_{\sigma}\ s)\ \mathit{allows}_{\sigma}\ I, \\
& \quad \exists O \in \mathcal{O}^{\sigma}_{I}(\mathcal{C}, s) \text{ such that } O - \overline{c}(I) \subseteq D.
\end{aligned}$$

Theorem 3.22. *Let* Γ, Δ *be single-level environments and suppose* $\Delta \vdash P, Q$, *where* Γ *is compatible with* Δ. *Then* $\Gamma \triangleright_{\sigma} P \sqsubseteq_{must} Q$ *if and only if* $\Gamma; \Delta \triangleright P \ll^{\sigma} \Gamma; \Delta \triangleright Q$.

The remainder of this subsection is devoted to the proof of this theorem. We will assume all triples $\Gamma; \Delta \triangleright P$ are configurations, and that all environments are single-level.

Proposition 3.23. $\Gamma; \Delta \triangleright P \ll^{\sigma} \Gamma; \Delta \triangleright Q$ *implies* $\Gamma \triangleright_{\sigma} P \sqsubseteq_{must} Q$.

Proof. (Outline) The proof follows the outline of that of Lemma 4.4.13 of [10], although the details are more complicated because of asynchrony and the use of type environments and security levels.

Let T be an arbitrary $\mathsf{new}\mathit{free}$ test such that $\Gamma \vdash_{\sigma} P$ and suppose P **must** T. We show Q **must** T. Let

$$T \mid Q \ (\equiv T_0 \mid Q_0) \xrightarrow{\tau} \mathcal{C}_1 \xrightarrow{\tau} \ldots\ldots \mathcal{C}_k \xrightarrow{\tau} \ldots \qquad (\dagger)$$

be an arbitrary maximal computation from $T \mid Q$, where we may assume each $\mathcal{C}_k$ has the form $(\mathsf{new}\, \tilde{c_k} : \tilde{C_k})\ T_k \mid Q_k$. We must show that for some k, $T_k \xrightarrow{\omega!\langle\rangle}$.

First suppose that the computation $(\dagger)$ is finite, ending in $\mathcal{C}_n$. Using Trace Decomposition it can decomposed into

$$\begin{aligned}
\Gamma; \Delta \triangleright Q &\overset{s}{\Longrightarrow} \Gamma; \Delta \triangleright Q_n \\
T &\overset{\overline{s}}{\Longrightarrow} T_n
\end{aligned}$$

From Lemma 2.5 we can assume T_n has the form $a_1!\langle v_1\rangle \mid \ldots a_k!\langle v_k\rangle \mid T'$, where T' cannot perform any input moves. Let I denote the multi-set of input actions, $\{a_1?v_1, \ldots, a_k?v_k\}$. We will use $\overline{I'} \mid T'$, where I' is a subset of I, to denote the the obvious term consisting of T' in parallel with the multi-set of output atoms determined by I'. Finally let D denote the acceptance set determined by the configuration $\mathcal{C}_n$. Note that $c(I) \cap D = \emptyset$.

At this stage let us suppose that $\Gamma; \Delta \triangleright P \Downarrow^{\sigma} s$. Then we can apply the hypothesis to obtain an $O \in \mathcal{O}^{\sigma}_{I}(\Gamma; \Delta \triangleright P, s)$ such that $O - \overline{c}(I) \subseteq D$. This means that there is a trace

$$\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}_\sigma \Gamma_1; \Delta_1 \triangleright P_1 \overset{I'}{\Longrightarrow}_\sigma \Gamma_m; \Delta_m \triangleright P_m$$

where

$$\Gamma_m; \Delta_m \triangleright P_m \not\xrightarrow{\tau} \quad (1)$$
$$\Gamma_m; \Delta_m \triangleright P_m \not\xrightarrow{a?v}_\sigma \text{ for any } \ a?v \in I - I' \quad (2)$$

By trace composition we can form

$$T \mid P \xrightarrow{\tau}{}^* T_n \mid P_1 \xrightarrow{\tau}{}^* (\overline{I_2} \mid T') \mid P_m, \ \ I_2 = I - I'.$$

If we can show that this is maximal, that is $(\overline{I_2} \mid T') \mid P_m \not\xrightarrow{\tau}$, then we are finished because P **must** T means that for some k, $T_k \xrightarrow{\omega!\langle\rangle}$.

The only possibility is a communication between P_m and $I_2 \mid T'$. In both cases below we rely on the fact that the environments are singl-level, enabling us to employ Lemmas 3.6 and 3.11.

Input: For some $a?v \in I_2, P_m \xrightarrow{a?v}$.
Here from Lemma 3.6 we know that Γ_1 and Γ_m are Γ *after*$_\sigma$ s. Applying Lemma 3.11 it follows that $\Gamma_m \vdash_\sigma a!\langle v\rangle$, which by Lemma 3.5 is sufficient to ensure that $\Gamma_m; \Delta_m \triangleright P_m \xrightarrow{a?v}_\sigma$. This contradicts (2) above.
Output: Here we have $P_m \xrightarrow{(c)a!\langle v\rangle}$ and $T' \xrightarrow{a?v}$.
Again from Lemma 3.11 we know $\Gamma_m \vdash_\sigma T'$ and therefore $a! \in O \subseteq D$; so $Q_n \xrightarrow{(\tilde{c})a!w}$ for some value w. Because of the structure of our language, $T' \xrightarrow{a?v}$ implies that $T' \xrightarrow{a?w}$ is also true, and therefore we have a contradiction of the maximality of $\mathcal{C}_n$.
This completes the proof, under the assumptions that $\Gamma; \Delta \triangleright P \Downarrow^\sigma s$ and the computation under scrutiny, (†), is finite. However these assumptions can be taken care of in the standard manner, as in the proof of Lemma 4.4.13 of [10].

□

As in the case of *may testing* the proof of the converse depends on the ability to define well-typed tests which determine the relation $\ll^\sigma$. Here there are two possible reasons why configurations may not be related; one associated with convergence, the other with a mismatch of acceptance sets. We treat each in turn. As in the previous sub-section to avoid notational complexity we only consider simple output actions, where only single names are transmitted. We also use some of the derived notation developed in that sub-section.

Tests for Convergence. We define the terms $T_C(\Gamma, s, \sigma)$ by induction on s:

ε**:** Here $T_C(\Gamma, s, \sigma) = \omega!\langle\rangle \oplus \omega!\langle\rangle$
$a!v \cdot s'$**:** Here $T_C(\Gamma, s, \sigma)$ is given by

$$\begin{aligned}(\mathsf{new}\, n)\;\; n!\langle\rangle \mid n?()\, \omega!\langle\rangle \mid a?(x : \mathrm{A})\; & \mathsf{if}\ x = v \\ & \mathsf{then}\ n?()\, T_C(\Gamma \sqcap v : \mathrm{A}, s', \sigma) \\ & \mathsf{else}\ \mathbf{0}\end{aligned}$$

where $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)$ for some $\delta \preceq \sigma$
$(c)a!c \cdot s'$**:** In this case $T_C(\Gamma, s, \sigma)$ is given by

$$\begin{aligned}(\mathsf{new}\, n)\;\; n!\langle\rangle \mid n?()\, \omega!\langle\rangle \mid a?(x : \mathrm{A})\; & \mathsf{if}\ x \in I(\Gamma, \mathrm{A}) \\ & \mathsf{then}\ \mathbf{0} \\ & \mathsf{else}\ (n?()\, T_C(\Gamma, c : \mathrm{A}, s', \sigma))\{\!|x/c|\!\}\end{aligned}$$

where again $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)$ for some $\delta \preceq \sigma$
$a?v \cdot s'$**:** Here $T_C(\Gamma, s, \sigma)$ is only defined if $\Gamma \vdash a : \mathsf{w}_\delta\langle \mathrm{A}\rangle, v : \mathrm{A}$ for some $\delta \preceq \sigma$, in which case it is

$$a!\langle v\rangle \mid T_C(\Gamma, s', \sigma)$$

We leave the reader to check the following:

Lemma 3.24. *If* $\Gamma; \Delta \triangleright Q \overset{s}{\Longrightarrow}_\sigma \Gamma'; \Delta' \triangleright Q'$, *where* $Q' \not\Downarrow$ *then*

- $T_C(\Gamma, s, \sigma)$ *is defined*
- $\Gamma \vDash_\sigma T_C(\Gamma, s, \sigma)$
- Q $\cancel{\textbf{must}}$ $T_C(\Gamma, s, \sigma)$.

Proof. By induction on s. □

Corollary 3.25. $\Gamma \triangleright_\sigma P \sqsubseteq_{must} Q$ *and* $\Gamma; \Delta \triangleright P \Downarrow^\sigma s$ *implies* $\Gamma; \Delta \triangleright Q \Downarrow^\sigma s$

Proof. Suppose, on the contrary, that for some s, $\Gamma; \Delta \triangleright P \Downarrow^\sigma s$, while $\Gamma; \Delta \triangleright Q \overset{s}{\Longrightarrow}_\sigma \Gamma'; \Delta' \triangleright Q$, for some Q' such that $Q' \not\Downarrow$. By the previous Lemma it is sufficient to show P **must** $T_C(\Gamma, s, \sigma)$, which can easily be done by induction on s. □

Tests for Acceptance Sets. Let us first extend the predicate $allows_\sigma$ to apply to output acceptance sets, in addition to sets of input actions. We write Γ $allows_\sigma$ O if, for each $a! \in O$, $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)$ for some $\delta \preceq \sigma$, and $\Gamma \vdash v : \mathrm{A}$ for some value v; note that this means $\Gamma \vDash_\sigma a!\langle v\rangle$.

We now define terms $T(\Gamma, s, O, I, \sigma)$, where O is an output acceptance set and I is a set of input actions, by induction on s. The inductive cases are very similar to the corresponding cases in the definition of the tests for convergence.

ε**:** Here $T(\Gamma, s, O, I, \sigma)$ is only defined if Γ *allows*$_\sigma$ O, I, in which case it is

$$\prod\{\, a!\langle v\rangle \mid a?v \in I \,\} \mid \prod\{\, a?(x : A_a)\, \omega!\langle\rangle \mid a! \in O \,\}.$$

Here the type A_a is determined by the fact that Γ *allows*$_\sigma$ O.

$a!v \cdot s'$**:** Here the test is given by

$$\begin{array}{ll}(\mathsf{new}\, n)\;\; n!\langle\rangle \mid n?()\,\omega!\langle\rangle \mid a?(x : \mathrm{A}) & \mathsf{if}\ x = v \\ & \mathsf{then}\ n?()\, T(\Gamma \sqcap v : \mathrm{A}, s', O, I, \sigma) \\ & \mathsf{else}\ \mathbf{0}\end{array}$$

where A is determined by $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)$ for some $\delta \preceq \sigma$.

$(c)a!c \cdot s'$**:** Here it is defined by

$$\begin{array}{ll}(\mathsf{new}\, n)\;\; n!\langle\rangle \mid n?()\,\omega!\langle\rangle \mid a?(x : \mathrm{A}) & \mathsf{if}\ x \in I(\Gamma, \mathrm{A}) \\ & \mathsf{then}\ \mathbf{0} \\ & \mathsf{else}\ (n?()\, T(\Gamma, c : \mathrm{A}, s', O, I, \sigma))\{\!|^{x}\!/\!_{c}|\!\}\end{array}$$

where, again, $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Gamma(a)$ for some $\delta \preceq \sigma$.

$a?v \cdot s'$**:** Here, as in the tests for convergence, the test is only defined if $\Gamma \vdash a : \mathsf{w}_\delta\langle \mathrm{A}\rangle, v : \mathrm{A}$ for some $\delta \preceq \sigma$, in which case it is

$$a!\langle v\rangle \mid T_C(\Gamma, s', O, I, \sigma)$$

We leave the reader to establish the following two Lemmas:

LEMMA 3.26. *Suppose* $(\Gamma\ \mathrm{after}_\sigma\ s)\ \mathrm{allows}_\sigma\ O, I$ *and that* $\Gamma; \Delta \triangleright Q \overset{s}{\Longrightarrow}_\sigma$*, for some* Δ*. Then* $T(\Gamma, s, O, I, \sigma)$ *is well-defined and* $\Gamma \vdash_\sigma T(\Gamma, s, O, I, \sigma)$*.* □

LEMMA 3.27. *Suppose* $T(\Gamma, s, O, I, \sigma)$ *is defined and* $O' \cap O \neq \emptyset$ *for every* $O' \in \mathcal{O}^\sigma_I(\Gamma; \Delta \triangleright P, s)$*. Then* $\Gamma; \Delta \triangleright P \Downarrow^\sigma s$ *implies* P ***must*** $T(\Gamma, s, O, I, \sigma)$*.*

We are now ready to prove the alternative characterisation:

THEOREM 3.28. *(Theorem 3.22)* $\Gamma \triangleright_\sigma P \sqsubseteq_{must} Q$ *if and only if* $\Gamma; \Delta \triangleright P \ll^\sigma \Gamma; \Delta \triangleright Q$

Proof. Because of the previous sequence of results it is sufficient to prove $\Gamma; \Delta \triangleright P \not\ll^\sigma \Gamma; \Delta \triangleright Q$ implies that there exists a test T such that $\Gamma \vdash_\sigma T$, P **must** T, while Q **m̸ust** T. In view of Corollary 3.25 there must be some s such that $\Gamma; \Delta \triangleright P \Downarrow^\sigma s$ and some computation

$$\Gamma; \Delta \triangleright Q \overset{s}{\Longrightarrow}_\sigma \mathcal{C}, \quad A = \mathcal{R}^\sigma(\mathcal{C}) \qquad (*)$$

and some I such that $(\Gamma\ \mathit{after}_\sigma\ s)\ \mathit{allows}_\sigma\ I$ and $c(I) \cap A = \emptyset$ with the property that for every $O' \in \mathcal{O}^\sigma_I(\Gamma; \Delta \triangleright P, s)$, $O' \not\subseteq A \cup \overline{c}(I)$. Let $\mathcal{O}^\sigma_I(\Gamma; \Delta \triangleright P, s) =$

$\{O_1, \ldots, O_n\}$ and for each i choose $a_i!$ such that $a_i! \in O_i - (A \cup \overline{c}(I))$. Let O be the set $\{a_1!, \ldots, a_n!\}$.

We now have all the ingredients to apply the previous two Lemmas to obtain the test $T(\Gamma, s, O, I, \sigma)$, well-typed with respect to Γ at level σ such that P **must** $T(\Gamma, s, O, I, \sigma)$. However the computation $(*)$ above shows that Q **mu$\not$st** $T(\Gamma, s, O, I, \sigma)$, since $\overline{c}(O) \cap c(I) = \emptyset$.

$\square$

4 Non-Interference Results

In this section we reconsider the approach taken to non-interference in Section 4 of [12]. The essential idea is that if a process is well-typed at a given level σ then its behaviour at that level is independent of processes "running at higher security levels"; or more generally "running at security levels independent to σ". A particular formulation of such a result was given in Theorem 5.3 of [12]:

THEOREM 4.1. *If* $\Gamma \vdash_{\sigma} P, Q$ *and* $\Gamma \vdash_{\top} H, K$, *where* H, K *are* σ-free *processes, then* $\Gamma \triangleright_\sigma P \simeq_{may} Q$ *implies* $\Gamma \triangleright_\sigma P \mid H \simeq_{may} Q \mid K$.

Here, because of our more refined notions of well-typing, we can give offer a significant improvement on this Theorem, and moreover the formulation is actually easier.

Let us say that the security level δ is independent of σ if $\delta \not\preceq \sigma$. We can ensure that a process H is "running at a security level independent to σ" by demanding that $\Delta \vdash^{\delta} H$, for some δ independent of σ. In fact we will only require the weaker typing relation $\Delta \vdash^{w\delta} H$. This ensures that all the output actions of H are at a level independent of σ, as can be deduced from the following property:

LEMMA 4.2. *Suppose* $\Delta \vdash^{w\delta} H$. *Then* $\Gamma; \Delta \triangleright H \xrightarrow{\mu}_\rho$, *where* μ *is an output action, implies* $\delta \preceq \rho$.

Proof. By induction on the derivation of $\Gamma; \Delta \triangleright H \xrightarrow{\mu}_\rho$. The only non-trivial case is the base case $\Gamma; \Delta \triangleright a!\langle v\rangle \xrightarrow{a!v}_\rho \mathbf{0}$.

Here we have $\Delta \vdash a : \mathsf{w}_{\delta'}\langle \mathrm{A}\rangle$ for some $\delta \preceq \delta'$. Because of (C-OUT) we know $\Gamma \vdash a : \mathsf{r}_{\rho'}\langle \mathrm{B}\rangle$ for some $\rho' \preceq \rho$. We can now apply Lemma 3.3 to obtain $\delta' \preceq \rho'$ from which it follows that $\delta \preceq \rho$. $\square$

We can now state our first non-interference result. Note that it applies to processes such that $\Delta \vdash_{r\sigma} P, Q$ rather than $\Delta \vdash_{\sigma} P, Q$; only their input actions need to be at level at most σ.

THEOREM 4.3 (NON-INTERFERENCE 1). *Let Γ and Δ be compatible and suppose $\Delta \vdash_{r\sigma} P, Q$. Then*

$$\Gamma \triangleright^{\sigma} P \lesssim_{may} Q \textit{ implies } \Gamma \triangleright^{\sigma} P \mid H \lesssim_{may} Q \mid K$$

provided $\Delta \vdash^{w\delta} H, K$ for some δ independent of σ.

Proof. Because of Proposition 3.13 it is sufficient to prove

$$\Gamma; \Delta \triangleright P \mid H \overset{s}{\Longrightarrow}_{\sigma} \quad \text{implies} \quad \Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}_{\sigma}$$

This is proved by induction on the derivation of $\Gamma; \Delta \triangleright P \mid H \overset{s}{\Longrightarrow}_{\sigma}$. The base case, when s is ε, is trivial, and there are three possibilities for the inductive case.

First suppose the derivation has the form

$$\Gamma; \Delta \triangleright P \mid H \xrightarrow{\alpha}_{\sigma} \Gamma'; \Delta' \triangleright R \overset{s'}{\Longrightarrow}_{\sigma}$$

Here there are two cases.

α is performed by P: So R has the form $P' \mid H$ and

$$\Gamma; \Delta \triangleright P \xrightarrow{\alpha}_{\sigma} \Gamma'; \Delta' \triangleright P'.$$

By Subject Reduction, Theorem 2.6, we know $\Delta' \vdash_{r\sigma} P'$ and therefore we can apply induction to obtain the result.

α is performed by H: Here R has the form $P \mid H'$ and

$$\Gamma; \Delta \triangleright H \xrightarrow{\alpha}_{\sigma} \Gamma'; \Delta' \triangleright H'.$$

From the previous Lemma we know α must be an input, say $a?v$, and from Lemma 3.2 we know that Γ' is simply Γ and Δ' must take the form $\Delta \sqcap v : \mathrm{A}$ for some type A. By weakening we therefore have $\Delta' \vdash_{r\sigma} P$ and we may apply induction to obtain $\Gamma'; \Delta' \triangleright P \overset{s'}{\Longrightarrow}_{\sigma}$.

From the same Lemma we know that $\Gamma \vdash a : \mathsf{w}_{\delta}\langle \mathrm{B} \rangle,\ v : \mathrm{B}$ for some $\delta \preceq \sigma$ and B <: A. So we can infer

$$\Gamma; \Delta' \sqcap v : \mathrm{B} \sqcap a : \mathsf{w}_{\delta}\langle \mathrm{B} \rangle \triangleright P \mid a!\langle v \rangle \overset{s'}{\Longrightarrow}_{\sigma} .$$

An application of (TR-ASYNC) now gives the required $\Gamma; \Delta \triangleright P \overset{\alpha \cdot s'}{\Longrightarrow}_{\sigma}$.

The second possibility is that the derivation is derived using an instance of (TR-ASYNC). Here a simple inductive argument suffices.

The final possibility is that it has the form

$$\Gamma; \Delta \triangleright P \mid H \xrightarrow{\tau}_{\sigma} \Gamma'; \Delta' \triangleright R \overset{s}{\Longrightarrow}_{\sigma}$$

If the initial τ action is performed either by P, or by H then (by Subject Reduction) we can apply induction to obtain the result. So there remains two cases:

Output from H to P: It turns out that this is not possible, because $\delta \not\preceq \sigma$.
Suppose we did have such an output. Then we would have

$$\Delta \models^{w\delta} H, \quad H \xrightarrow{(\tilde{c})a!v} H'$$
$$\Delta \vdash_{r\sigma} P, \quad P \xrightarrow{a?v} P'$$

Applying Subject Reduction we would have

$$\Delta \vdash a : \mathsf{w}_{\delta'}\langle \mathrm{A} \rangle, \quad \delta \preceq \delta'$$
$$\Delta \vdash a : \mathsf{r}_{\sigma'}\langle \mathrm{B} \rangle, \quad \sigma' \preceq \sigma.$$

The consistency requirement on types implies $\delta' \preceq \sigma'$, which contradicts $\delta \not\preceq \sigma$.

Output from P to H: Here the derivation takes the form

$$\Gamma; \Delta \triangleright P \mid H \xrightarrow{\tau}_{\sigma} \Gamma; \Delta \triangleright (\tilde{c} : \tilde{C})(P' \mid H') \overset{s}{\Longrightarrow}_{\sigma}$$

where $P \xrightarrow{(\tilde{c})a!v} P'$ and $H \xrightarrow{a?v} H'$. So there exists a sequence s_C, associated with s, such that

$$\Gamma; \Delta\, ; \tilde{c} : \tilde{C} \triangleright P' \mid H' \overset{s_C}{\Longrightarrow}_{\sigma} \qquad (*)$$

with the property that for for any R such that $\Gamma; \Delta\, ; \tilde{c} : \tilde{C} \triangleright R \overset{s_C}{\Longrightarrow}_{\sigma}$ it follows that $\Gamma; \Delta \triangleright (\tilde{c} : \tilde{C})R \overset{s}{\Longrightarrow}_{\sigma}$.

Applying induction to $(*)$ we obtain

$$\Gamma; \Delta\, ; \tilde{c} : \tilde{C} \triangleright P' \overset{s_C}{\Longrightarrow}_{\sigma}$$

Note that this is possible since Subject Reduction gives

$$\Delta, \tilde{c} : \tilde{C} \vdash_{r\sigma} P', \quad \Delta \sqcap v : \mathrm{A} \models^{w\delta} H'$$

where A is a type such that $\Delta, \tilde{c} : \tilde{C} <: \Delta \sqcap v : \mathrm{A}$. (In fact A is the type at which v is sent by P.)

It follows that $\Gamma; \Delta, \tilde{c} : \tilde{C} \triangleright P' \mid a!\langle v \rangle \overset{s_C}{\Longrightarrow}_{\sigma}$ and therefore

$$\Gamma; \Delta \triangleright (\mathsf{new}\, \tilde{c} : \tilde{C})\ (P' \mid a!\langle v \rangle) \overset{s}{\Longrightarrow}_{\sigma}\ .$$

But by Lemma 2.5 we know

$$P \equiv (\mathsf{new}\, \tilde{c} : \tilde{C})\ (P' \mid a!\langle v \rangle)$$

and the result follows.

□

We end the paper with a non-interference result with respect to *must* testing. Note that Theorem 4.3 is no longer true when $\sqsubseteq_{may}$ is replaced by $\sqsubseteq_{must}$, as the following example shows.

EXAMPLE 4.4. Let A denote the type $\{\mathsf{w}_{\mathsf{bot}}\langle\rangle, \mathsf{r}_{\mathsf{bot}}\langle\rangle, \mathsf{r}_{\mathsf{top}}\langle\rangle\}$ and B denote $\{\mathsf{r}_{\mathsf{top}}\langle\rangle\}$. Further, let Γ map a to A and n to the type $\{\mathsf{w}_{\mathsf{bot}}\langle \mathrm{A}\rangle, \mathsf{r}_{\mathsf{bot}}\langle \mathrm{A}\rangle, \mathsf{r}_{\mathsf{top}}\langle \mathrm{B}\rangle\}$. Now consider the processes P and H defined by

$$P \Leftarrow n!\langle a\rangle \mid n?(x:\mathrm{A})\ x!\langle\rangle \qquad\qquad H \Leftarrow n?(x:\mathrm{B})\,\mathbf{0}$$

It is very easy to check that $\Gamma \vdash_{r\mathsf{bot}} P$ and $\Gamma \vdash^{w\mathsf{top}} H$. However

$$\Gamma;\Gamma \triangleright^{\mathsf{bot}} P \mid \mathbf{0} \not\sqsubseteq_{must} P \mid H$$

because of the bot level test $a?()\ \omega!\langle\rangle$.

The presence or absence of H determines whether or not there is read contention on the channel n, which in turn influences the deadlock capabilities of P with respect to the channel a.

Here the problem is the type of the channel n; it may be read at both level bot and top. Note that such examples, where there is contention between reads at different levels, can not be expressed in the *join calculus*, [5].

A not unreasonable restriction would be to require that the read capability of channels be confined to a particular security level. This would not rule out inter-level communication, but simply control it more tightly. This restriction can be enforced by requiring the type-checking to use single-level types and forbidding high-level processes to read from low-level channels.

THEOREM 4.5 (NON-INTERFERENCE 2). *Let Γ and Δ be compatible single-level environments and suppose $\Delta \vdash_{r\sigma} P, Q$. Then*

$$\Gamma \triangleright^{\sigma} P \sqsubseteq_{must} Q \textit{ implies } \Gamma \triangleright^{\sigma} P \mid H \sqsubseteq_{must} Q \mid K$$

for all finite processes H, K such that $\Delta \vdash^{\delta} H, K$ for some δ independent of σ.

Note that we must restrict our attention to finite H and K since *must* testing is sensitive to divergence; if H is a divergent term then we could not expect $\Gamma \triangleright^{\sigma} P \mid \mathbf{0} \simeq_{must} P \mid H$ to hold when P is a convergent term. This problem is avoided by restricting attention to finite terms,which can never diverge.

Restricting H and K to use channels at level at least δ, together with the use of single-level types, ensures that there is no contention, as exhibited in Example 4.4, between high and low level processes over read access to a channel.

The remainder of the section is devoted to the proof of this final result of the paper. Throughout we will assume Γ and Δ are compatible single-level environments, $\Delta \vdash_{r\sigma} P$, $\Delta \Vdash^{\delta} H$ for some δ independent of σ, and moreover that H is a finite process.

LEMMA 4.6. *For every s, $\Gamma; \Delta \triangleright P \Downarrow^{\sigma} s$ if and only if $\Gamma; \Delta \triangleright P \mid H \Downarrow^{\sigma} s$.*

Proof. One direction is easy, $\Gamma; \Delta \triangleright P \not\Downarrow^{\sigma} s$ implies $\Gamma; \Delta \triangleright P \mid H \not\Downarrow^{\sigma} s$.

Conversely, because H is finite, we can assume that

$$\Gamma; \Delta \triangleright P \mid H \overset{s}{\Longrightarrow}_{\sigma} (\tilde{c} : \tilde{C})(P' \mid H')$$

for some P' such that $P' \not\Downarrow$. We leave the reader to prove, by induction on this derivation, that $\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}_{\sigma} (\tilde{c'} : \tilde{C'})P'$ for some $\tilde{c'}, \tilde{C'}$. $\square$

PROPOSITION 4.7. *Suppose $A \in \mathcal{A}^{\sigma}(\Gamma; \Delta \triangleright P, s)$ and I is a multi-set of inputs such that $c(I) \cap A = \emptyset$ and $(\Gamma \mathrm{after}_{\sigma} s) \mathrm{allows}_{\sigma} I$. Suppose further that $\Gamma; \Delta \triangleright P \Downarrow^{\sigma} s$. Then there exists some $O \in \mathcal{O}^{\sigma}_{I}(\Gamma; \Delta \triangleright P \mid H, s)$ such that $O - \overline{c}(I) \subseteq A$.*

Proof. By induction on the derivation

$$\Gamma; \Delta \triangleright P \overset{s}{\Longrightarrow}_{\sigma} \mathcal{D}, \quad \text{where } A = \mathcal{R}^{\sigma}(\mathcal{D})$$

- The empty derivation.
  Here $A = \mathcal{R}^{\sigma}(\Gamma; \Delta \triangleright P)$. This means that $P \overset{\tau}{\not\longrightarrow}$ but we may have $P \mid H \overset{\tau}{\longrightarrow}$ either because $H \overset{\tau}{\longrightarrow}$ or there maybe a write up from P to H. But because H is syntactically finite and $P \Downarrow$ we know there is some $P' \mid H'$ such that $P \mid H \overset{\tau}{\longrightarrow}^{*} P' \mid H' \overset{\tau}{\not\longrightarrow}$.
  By Lemma 2.5 we know that $\mathcal{R}^{\sigma}(\Gamma; \Delta \triangleright P' \mid H') \subseteq A$ and therefore, since $c(I) \cap A = \emptyset$, $\Gamma; \Delta \triangleright P' \mid H' \searrow^{\sigma}_{I} \Gamma; \Delta \triangleright P' \mid H'$. The required O may therefore be taken to be the output subset of A.
- The derivation has the form $\Gamma; \Delta \triangleright P \xrightarrow{(c)a!v}_{\sigma} \Gamma'; \Delta' \triangleright P' \overset{s}{\Longrightarrow}_{\sigma} \mathcal{D}$.
  By Subject Reduction we know $\Delta' \vdash_{r\sigma} P'$ and therefore we may apply induction to obtain $O \in \mathcal{O}^{\sigma}_{I}(\Gamma; \Delta \triangleright P' \mid H, s)$ with the required properties. The result now follows since $\mathcal{O}^{\sigma}_{I}(\Gamma; \Delta \triangleright P' \mid H, s) \subseteq \mathcal{O}^{\sigma}_{I}(\Gamma; \Delta \triangleright P \mid H, (c)a!v \cdot s)$
- The remaining cases are similar.

$\square$

We also have the converse.

PROPOSITION 4.8. *Suppose $A \in \mathcal{A}^\sigma(\Gamma; \Delta \triangleright P \mid H, s)$ and, as in the previous Proposition, I is a set of inputs such that $c(I) \cap A = \emptyset$ and $(\Gamma \,\mathrm{after}_\sigma s)\,\mathrm{allows}_\sigma I$. Then there exists some $O \in \mathcal{O}^\sigma_I(\Gamma; \Delta \triangleright P, s)$ such that $O - \overline{c}(I) \subseteq A$.*

Proof. Again by induction on the derivation

$$\Gamma; \Delta \triangleright P \mid H \overset{s}{\Longrightarrow}_\sigma \mathcal{D}, \quad \text{where } A = \mathcal{R}^\sigma(\mathcal{D})$$

As an example we examine the case

$$\Gamma; \Delta \triangleright P \mid H \xrightarrow{\tau} \mathcal{D}' \overset{s}{\Longrightarrow}_\sigma \mathcal{D},$$

where the initial τ consists of a communication between P and H. This must be a write-up from P to H; so $\mathcal{D}'$ has the form $\Gamma; \Delta \triangleright (\tilde{c}\!:\!\tilde{C})P' \mid H'$, where $P \xrightarrow{(\tilde{c}:\tilde{C})a!v} P'$ and $H \xrightarrow{a?v} H'$. We know P has the form $(\tilde{c}\!:\!\tilde{C})(a!\langle v\rangle \mid P')$, but more importantly that $\mathsf{r}_\delta\langle \mathrm{A}\rangle \in \Delta(a)$ for some δ independent from σ (†). What this means is there can can be no communication between $a!\langle v\rangle$ and any Q such that $\Delta \vdash_{r\sigma} Q$.

Now the derivation $\Gamma; \Delta \triangleright (\tilde{c}\!:\!\tilde{C})P' \mid H' \overset{s}{\Longrightarrow}_\sigma \mathcal{D}$ can be transformed into $\Gamma; \Delta, \tilde{c}\!:\!\tilde{C} \triangleright P' \mid H' \overset{s_C}{\Longrightarrow}_\sigma \mathcal{E}$, where $\mathcal{R}^\sigma(\mathcal{D}) = \mathcal{R}^\sigma(\mathcal{D})$. Moreover we can apply induction to this derivation, to obtain $O \in \mathcal{O}^\sigma_I(\Gamma; \Delta, \tilde{c}\!:\!\tilde{C} \triangleright P', s_C)$ such that $O - \overline{c}(I) \subseteq A$.

We can use (†) to prove O is also in $\mathcal{O}^\sigma_I(\Gamma; \Delta, \tilde{c}\!:\!\tilde{C} \triangleright a!\langle a\rangle \mid P', s_C)$. The result now follows since

$$\mathcal{O}^\sigma_I(\Gamma; \Delta, \tilde{c}\!:\!\tilde{C} \triangleright a!\langle a\rangle \mid P', s_C) \subseteq \mathcal{O}^\sigma_I(\Gamma; \Delta, \tilde{c}\!:\!\tilde{C} \triangleright \tilde{c}\!:\!\tilde{C}a!\langle a\rangle \mid P', s).$$

□

COROLLARY 4.9. *(Theorem 4.5) suppose $\Delta \vdash_{r\sigma} P, Q$. Then*

$$\Gamma \triangleright^\sigma P \sqsubseteq_{must} Q \textit{ implies } \Gamma \triangleright^\sigma P \mid H \sqsubseteq_{must} Q \mid K$$

for all finite processes H, K such that $\Delta \vdash^\delta H, K$ for some δ independent of σ.

Proof. It is sufficient to prove

$$\Gamma; \Delta \triangleright P \ll^\sigma \Gamma; \Delta \triangleright P \mid H \text{ and } \Gamma; \Delta \triangleright P \mid H \ll^\sigma \Gamma; \Delta \triangleright P.$$

These follow from the two previous Propositions and Lemma 4.6. □

5 Conclusions and Related Work

This paper is a direct continuation of the research reported in [12]. There we focused on the general topic of security types, showing that *resource access control* could be enforced using a typing system and *information flow control* could be obtained by a restriction to the set of types employed. The import of Subject Reduction was emphasised by developing a Type Safety Theorem, which in turn required a version of the language in which processes were tagged with their security levels. Here we concentrated on types for *information flow*, calling the resulting language the security π-calculus. The first main result consists of alternative characterisations of *may* and *must* testing for this language. These use a novel labelled transition system, with judgements of the form

$$\Gamma; \Delta \triangleright P \xrightarrow{\mu}_{\sigma} \Gamma'; \Delta' \triangleright P'$$

which records the security levels at which actions occur, together with their effect on the type environment of the process under observation, Δ, and the effect on the, possibly different, type environment of the observing process, Γ. This labelled transition system is a generalisation of that used in [11] to characterise typed behavioural equivalences for the π-calculus. There judgements take the simpler form

$$\Gamma \triangleright P \xrightarrow{\mu} \Gamma' \triangleright P'$$

in which the type environment of the observed process, namely Δ, and changes to this environment, remain implicit. Our characterisation theorems would also be expressed in terms of these more abstract judgements, but the explicit use of the type environment of the observed processes, the Δs, makes the statement of our non-interference results more straightforward; see the formulations of Theorem 4.3 and Theorem 4.5.

Our second main result extends the non-interference result from [12], showing that non-interference, with respect to both *may* and *must* testing, can be enforced using types. However it remains to be seen to what extent this approach, non-interference through types, can be used to obtain useful instances of non-interference. For example in [8] a wide range of security properties have been shown to be expressible in terms of non-interference and it would be interesting to see whether these can be enforced by typing constraints using a type system such as ours. This would involve extending our language to include cryptographic primitives, such as those from [1], but we believe that this is not problematic.

A general overview of the use of static analysis techniques to enforce information-flow policies may be found in [22]. Useful surveys of research into non-interference in process languages are given in [6,21] [1] . Much of this work is behaviour based;

[1] For the use of types for other languages see [23].

systems are deemed to be interference-free if their trace sets, sequences of actions labelled *high* or *low*, satisfy certain properties. Here we use a more extensional approach, saying that a system is interference-free if low-level observers are unable to discern the presence or absence of high-level components. Such high-level definitions may be intuitively attractive but they are not necessarily easy to deal with. So, for example, to obtain our non-interference results we needed to give more intensional characterisations, in terms of sequences of high and low level actions. Moreover our use of these characterisations, in the proofs of our non-interference results, are very similar in nature some of the definitions of non-interference given in [6,7]. For example the proof technique used in Theorem 4.3 recalls the non-interference property called NDC in [6]. However a formal comparison is not straightforward; definitions, in papers such as [20,6,21] are for very simple untyped versions of CCS or CSP, while we deal with the more expressive π-calculus.
However the main difference in the two approaches may be summarised as follows:

- we propose validating a process for non-interference using syntax-directed typechecking
- in [6] and related work, a process is validated by checking semantics-based properties such as trace sets.

In [15] a type system is given which guarantees non-interference with respect to an extension of the π-calculus; moreover non-interference is expressed with respect to a barbed congruence. However the language used is a considerable extension of the π-calculus, with operators for selection based input/output, based on disjunctive patterns, and it is the behaviour of these operators which are mainly constrained by the type system. The types used are also very sophisticated. Unlike ours, which are simply annotated versions of the standard Pierce/Sangiorgi types, [18], they track the use of channels, using annotated affine and linear types, and capture causal relationships between actions by a partial composition on these types, using ideas based on the *graph types* of [25].

Finally [5], which uses security labels attached to messages in the join calculus to formulate non-interference, argues via an example for the use of a behavioural equivalence stronger than *may* testing. The formulation uses *weak barbed congruence* but could have equally well used *must* testing equivalence; indeed it is difficult to envisage a practical scenario in which there is something to be gained from assuming attackers have the extra power associated with the former rather than the latter. We have also already pointed out (in Example 4.4) that the join calculus can not be used to express situations in which there is read contention between different security levels. Nevertheless the approach used to develop a type system for the join calculus for detecting information flow seems to be quite general and may also be applicable to the asynchronous π-calculus.

References

[1] Martín Abadi. Secrecy by typing in security protocols. In *Proceedings of TACS'97*, volume 1281 of *Lecture Notes in Computer Science*, pages 611–637. Springer Verlag, 1997.

[2] D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical report MTR-2997, MITRE Corporation, 1975.

[3] G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA-Sophia Antipolis, 1992.

[4] Ilaria Castellani and Matthew Hennessy. Testing theories for asynchronous languages. In V Arvind and R Ramanujam, editors, *18th Conference on Foundations of Software Technology and Theoretical Computer Science (Chennai, India, December 17–19, 1998)*, LNCS 1530. Springer-Verlag, December 1998.

[5] Sylvain Conchon. Modular information flow analysis for process calculi. In Iliano Cervesato, editor, *Proceedings of the Foundations of Computer Security Workshop (FCS 2002)*, Copenhagen, Denmark, JULY 2002.

[6] Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1), 1995.

[7] Riccardo Focardi and Roberto Gorrieri. Non interference: Past, present and future. In *Proceedings of DARPA Workshop on Foundations for Secure Mobile Code*, 1997.

[8] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Non-interference for the analysis of cryptographic protocols. In U. Montanari and J. Rolim, editors, *27th Internationa Conference on Automata, Languages and Programming*, LNCS 1853, pages 354–371. Springer-Verlag, July 2000.

[9] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and privacy*, 1992.

[10] M. Hennessy. *An Algebraic Theory of Processes.* MIT Press, 1988.

[11] Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping (extended abstract). In James Harland, editor, *Electronic Notes in Theoretical Computer Science*, volume 61. Elsevier Science Publishers, 2002. Full version to appear in *Mathematical Structures in Computer Science.*

[12] Matthew Hennessy and James Riely. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Transactions on Programming Languages and Systems*, 24(5):566–591, September 2002.

[13] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[14] Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In P. Wegner M. Tokoro, O. Nierstrasz, editor, *Proceedings of the ECOOP '91 Workshop on Object-Based Concurrent Computing*, volume 612 of *LNCS 612*. Springer-Verlag, 1992.

[15] Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. In *29th Annual Symposium on Principles of Programming Languages*. ACM, January 2002.

[16] R. Milner, J. Parrow, and D. Walker. Mobile logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.

[17] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.

[18] B. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM*, 47(3):531–584, 2000.

[19] Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996. Extended abstract in LICS '93.

[20] A.W. Roscoe, J.C.P. Woodcock, and L. Wulf. Non-interference through determinism. In *European Symposium on Research in Computer Security*, volume 875 of *LNCS*, 1994.

[21] P.Y.A. Ryan and S.A. Schneider. Process algebra and non-interference. In *CSFW 12*. IEEE, 1997.

[22] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communication*, 2002. To appear.

[23] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998.

[24] David Turner. *The Polymorphic Pi-Calculus: Theory and Implementation.* Ph.d. thesis, Edinburgh University, 1995.

[25] Nobuko Yoshida. Graph types for monadic mobile processes. In *FSTTCS*, volume 1180, pages 371–386. Springer-Verlag, 1996.