DEPLOYING PROMELA/SPIN-BASED TEST GENERATION ON RTEMS: A PROGRESS REPORT

Andrew Butterfield

Lero - the Irish Software Research Centre,
Trinity College Dublin,
Andrew.Butterfield@scss.tcd.ie

Presenter: *Andrew Butterfield*

1. INTRODUCTION

As part of the ESA-sponsored activity "Qualification of RTEMS Symmetric Multiprocessing (Space Profile RTEMS)", from Nov 2018–Dec. 2021, a technique was developed to use the Promela modelling language to construct formal models of key parts of RTEMS (Real-Time Executive for Multiprocessor Systems), and the SPIN model-checker to support test generation from those models. The activity was led by Thales Edisoft (PT), and involved embedded brains GmbH (DE), Jena Optronik GmbH(DE), CISTER at Instituto Superior de Engenharia do Porto (PT), and ourselves at Lero, the Irish Software Research Centre, at Trinity College Dublin (IE).

Ours was one contribution to an activity whose overall purpose was to have a toolkit that generates all verification and validation evidence as documents that satisfy the ECSS-E-ST-40C [ECS09] and ECSS-Q-ST-80C [ECS17] standards, along with datapacks of such evidence for target configurations, to criticality level C. An important goal was that all this tooling, and the datapack, would be fed back to the RTEMS open-source repositories themselves. This is part of a long term goal to have qualification processes made a standard part of RTEMS software engineering principles.

A follow-up activity, "Independent Software Verification and Validation (ISVV) of the RTEMS SMP Qualification Data Pack", in 2022, run by Critical Systems (PT), took this work to criticality level B. This included reviewing extra material we produced after the first activity was officially over.

To summarise, the overall goal has been to prequalify the multicore version of RTEMS to ECSS Category C & B. Key aspects where:

1. Develop tools and *specification* data to automate testing, traceability and most of the reporting.
2. Focus on LEON processors GR712rc and GR740, using Classic API subset ("Space Profile").
3. Make all results *available open-source* through RTEMS
4. Take the opportunity to explore a role for *Formal Methods*.

2. METHODOLOGY

2.1. Formal Methods

Formal Methods are techniques that use mathematical models of requirements, specifications, and implementations, to rigorously demonstrate correctness. The motivation is that formal reasoning, based on calculation and proof, can cover all possibilities, including infinite behaviors, in a way that increases confidence. This is feasible today because of the availability of powerful tool support. The key barrier to wider adoption is the level of skill and mathematical knowledge required by the use to have such confidence.

2.2. Model Checking

There is a sub-genre of formal Methods known as model-checking. These are based of formal languages that are more programmatic in style, and which have a semantics based on finite-state machines (models). Model-checkers take such models and check that specified properties hold by doing an exhaustive search from an initial state. If no errors are found, that is simply reported. However, if an error is discovered, it is reported, along with a counter-example — a series of steps in the model that leads to the point of failure. An engineer can easily understand

such counter-examples, and map them to the real system being constructed. This has made model-checking the most widely used formal technique in industry[HBB+09]. A simple trick then makes it very easy to use an error-free model, once obtained, to generate tests for that real system.

2.3. Getting Tests from a Model-Checker

We used the SPIN model-checker[Hol04] for test generation. The SPIN model-checker takes a model written in Promela, and performs a range of analyses on it. These range from checking general properties, such as deadlock- and livelock-freedom, to application specific asssertions ans temporal properties. A check either confirms that a property holds, or, should it fail, provides a concrete scenario that shows how the failure arises.

Given a model whose desirable properties have all been verified, our approach to test generation is, for each such property individually : replace that property by its negation, and re-run the model-checker. This will result in the generation of counter-examples that show *correct* behaviour. These are then taken as specifications of precise, deterministic, reproducible test cases.

These are then mapped (refined) to actual test code. This is achieved by the addition of a refinement file that maps basic model actions and events to corresponding test program code.

2.4. Why Promela/SPIN?

The long-term objective was that the formal methods material we developed would be made available to the RTEMS community, ideally as part of their repositories. After exploring a range of formal techniques, with a strong focus on Frama-C, Isabelle/HOL, TLA+, and Promela/SPIN, we elected to adopt the latter for the formal verification effort. Promela/SPIN[Hol04] was judged to be both an effective technique, recommended by others (*e.g.*, [HBB+09]), as well as one that would fit well with the RTEMS community guidelines (*e.g.*, open-source, reproducible builds, wide host support, maintainability).

3. TEST GENERATION PROCESS

The test generation process for a given function, API, or RTEMS Manager, consists of the following steps:

1. Develop and verify a Promela model of correct behaviour.
2. Develop a mapping from observable Promela behavior to test code.
3. Use python scripts we developed to automate the test generation and deployment.

3.1. RTEMS Models

The following four models are currently present in `rtems-central`: the Chains API, and the Event, Barrier, and Message Managers. These models are found in `formal/promela/models`. A newer model has been developed of part of the Semaphore Manager, but currently is only available via `https://github.com/andrewbutterfield/RTEMS-SMP-Formal`.

We give a brief summary of each below. We do a deeper dive into the Barrier model.

3.1.1. Chains API (`formal/promela/model/chains`)

We used a model of the unprotected variants of chain append and remove [RTEa, §34] to work out initial ideas. This involved developing the basic mechanisms to get useful output from SPIN, and specifying how that gets mapped to RTEMS test code. The other 21 API calls were not modelled. The main limitation of this work was that it did not address concurrency issues.

3.1.2. The Event Manager (`formal/promela/model/events`)

During the ESTEC activity, a model of most of the Event Manager [RTEa, §15] features was developed. It built on the chains work and added ways to manage concurrency (tasks, priority, preemption, multiple cores). It modelled both of the API calls (send/receive). The model was also extended to cover all the features as part of Masters level dissertation [Jen21]. This overlapped with the end of the ESTEC activity.

3.1.3. The Barrier Manager (`formal/promela/model/barriers`)

A model of the Barrier Manager [RTEa, §13] was developed as a Masters dissertation [Jaś22]. It covered all 5 API calls (create/ident/delete/wait/release). It followed the basic design and structure of the Events Manager model. It was included in the follow-up ESTEC IV&V activity in 2022.

3.1.4. The Message Manager (`formal/promela/model/messages`)

A model of the Message Manager [RTEa, §14] was developed as a Masters dissertation [Lyn22]. It covered 3 of the 11 API calls (create/send/receive). It also followed the basic design and structure of the Events Manager model, and was also included in the follow-up ESTEC IV&V activity in 2022.

3.1.5. The Semaphore Manager (`formal/promela/model/semaphores`)

A model of part of the Semaphore Manager [RTEa, §12] was developed as a Masters dissertation [Fla23]. It focused on counting and binary semaphores using a FIFO queuing policy (create/ident/delete/obtain/release/flush).

3.2. The Barrier Manager Model

The overall model structure has some initial setup, and then `run`s five Promela processes. Three of these, `Runner`, `Worker0`, and `Worker1`, model the three RTEMS Tasks that will be used to implement the tests. The other two, `System`, and `Clock` model relevant scheduler and timer behaviour.

Each API call has its own model, that captures its behaviour, based on a careful reading of the relevant RTEMS documentation.

The basic idea behind all these models is that the runner and worker processes are each going to make some sequence of API calls with some variations of parameters and options. Part of the setup is `chooseScenario` which exploits the explicit non-determinism in Promela to randomly choose the sequencing and options:

```
mtype = {ManAcqRel,AutoAcq,AutoToutDel};
mtype scenario;
inline chooseScenario() {
  if // non-deterministically pick one
  ::  scenario = ManAcqRel;
  ::  scenario = AutoAcq;
  ::  scenario = AutoToutDel;
  fi
  if // 3 way conditional
  ::  scenario == ManAcqRel ->
        // setup manual acq.-rel.
  ::  scenario == AutoToutDel -> ...
  ::  scenario == AutoAcq ->  ...
  fi
}
```

When test generation is invoked, looking for all test counter-examples, all those choices get exercised. The result is one test for every possible sequencing/interleaving of behavior defined in the model.

3.3. Mapping Promela Behavior to C Test Code

While SPIN outputs counter-examples that are easy for the modeller to read and interpret (given access to the model sources), they are not suitable for parsing and analysing. Instead, the modeller uses the Promela `printf` statement to output important observations in a simple machine-readable format. Such observations will include the values of variables, as well as documenting API calls and their parameter values. A sample of the barrier model output is:

```
@@@ 3 CALL barrier_create 0 0 0 1
@@@ 3 SCALAR rc 3
```

The mappings from observations to C test-code snippets are defined in refinement files (`*-rfn.yml`) that contain YAML dictionaries. The `spin2test` program takes the `CALL barrier_create 0 0 0 1`, and looks up `barrier_create` in the refinement file. The following is a simplified version of that entry:

```
barrier_create: |
  attr = mergeattribs({1});
  rc = rtems_barrier_create
         ({0},attr,{2},{3}?&bid:NULL);
```

The parameters `0 0 0 1` are substituted in to get the following C test code snippet:

```
attr = mergeattribs(0);
rc = rtems_barrier_create
       (0,attr,0,1?&bid:NULL);
```

The number immediately after the `@@@` is the relevant Promela process identifier, and is used to ensure the C code ends up in the body of the corresponding RTEMS test Task, and helps with generating code for test synchronisation.

3.4. Test Generation Support Software

The ESTEC activity delivered a range of python tools to perform the test generation process. These can be found in `formal/promela/src`.

The key component was a program `spin2test` which would take *one* test scenario and generate a single test runner (Fig. 1). It converts one SPIN counterexample (`*.spn`) into a single RTEMS Test Framework test runner (`tr-*.c`). The Promela to C code mapping is defined by a refinement file (`*-rfn.yml`). The

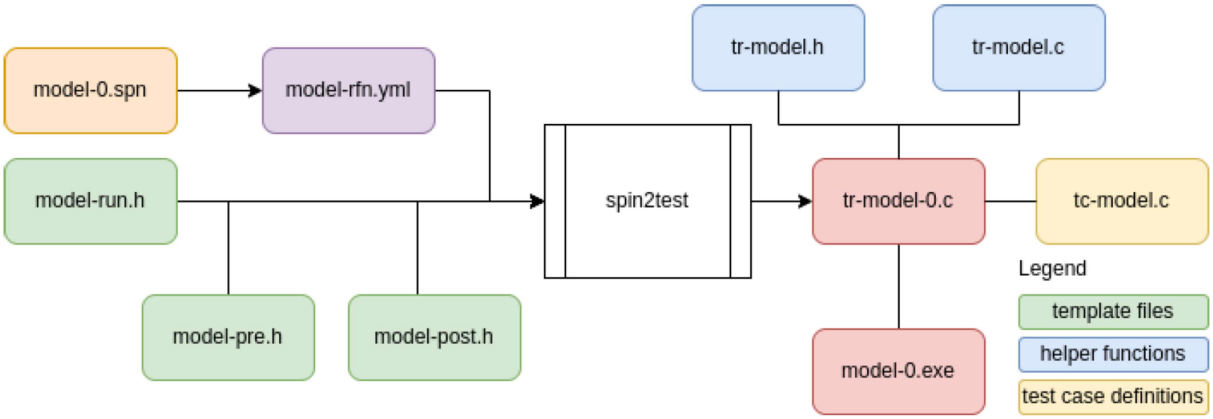

Figure 1. `spin2test` overview [Lyn22].

program would have to be run individually for each such scenario.

The use of automation to run this is provided by the program `testbuilder`, originally developed by the student who extended the coverage of the Events Manager [Jen21]. Given a model, it runs the model to generate all the counter-examples. For each counter-example, it runs `spin2test` to generate the test-code, deploys it to the relevant RTEMS test sources directory, builds the tests, runs them and save the output in a logfile (Fig. 2). More recent work by a student raised the level of automation in `testbuilder` to the extent that all available models can be generated and run with just one command-line invocation[Hun23]

A current Masters-level project is looking at further improvements to the `testbuilder` software, and some early improvements from this work are already available.

3.5. Formal Verification documentation

At the time of writing, documentation for this work is in the process of being added into the RTEMS Software Engineering guide [RTEb]. What has been proposed is a new Software Verification chapter just after the material on the Software Test Framework. This is following the RTEMS standard approach to adding new material using `git` patch sets.

4. CONCLUSION

Key progress has been made in using Promela/SPIN to produce models of various aspects of RTEMS. These models can be used to generate tests, and to check various desirable properties of systems. The range and coverage of these models will continue to grow over time, as a result of current and future academic research activity, and possibly as part of further joint activities in this area.

4.1. State of Play (July 2023)

Currently available:

QDP (`(https://rtems-qual.io.esa.int/)`
C Test code only: Chains API; Event Manager

RTEMS (`https://git.rtems.org/rtems-central/tree/`)

Models `../tree/formal/promela/models/`
Chains API; Event, Barrier and Message Managers

Software `../tree/formal/promela/src/`
spin2test; testbuilder

Under development for RTEMS:

Models Semaphore Manager; SMP thread-queue models.

Software testbuilder automation upgrades

C Test Code Ready to deploy to RTEMS, under review.

Documentation New "Formal Verification" section for the RTEMS Software Engineering Manual, under review.

The "bleeding edge" of this work can be found on Github at `https://github.com/andrewbutterfield/RTEMS-SMP-Formal`.

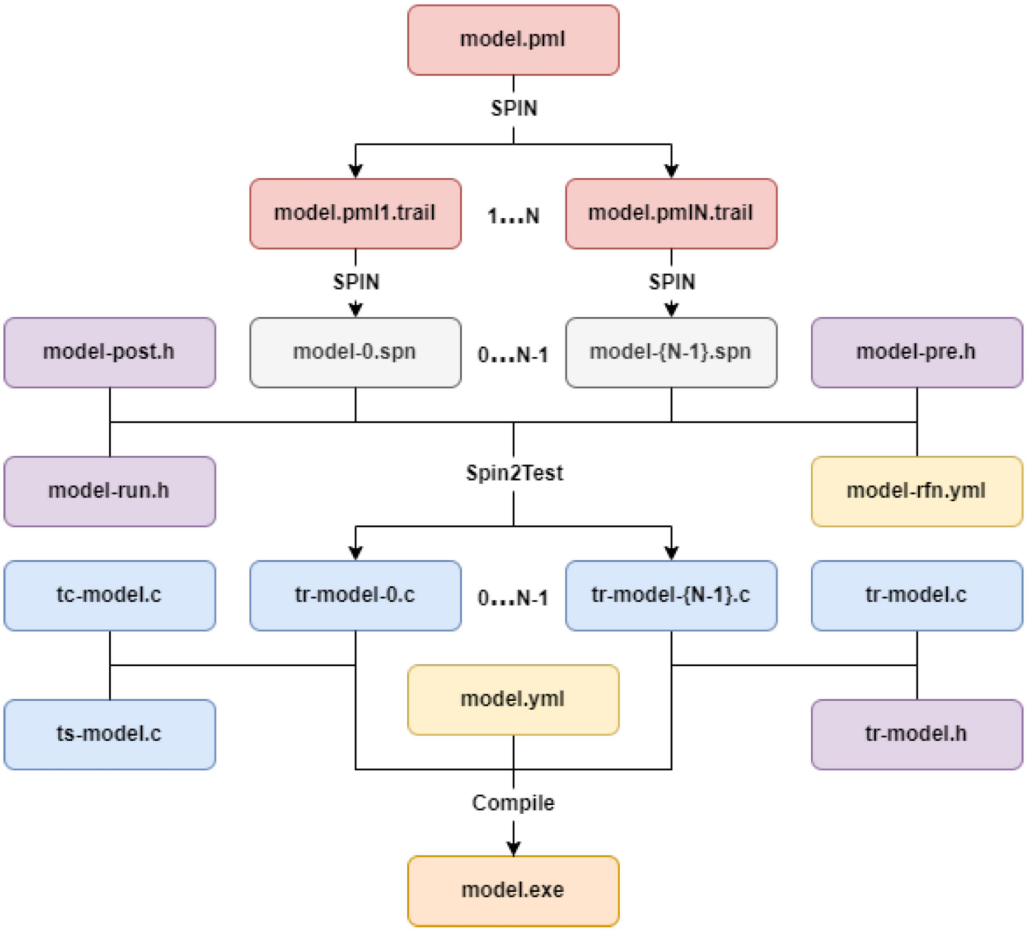

Figure 2. `testbuilder` *overview [Jaś22]*

REFERENCES

[ECS09] ECSS. *ECSS-E-ST-40C - Software general requirements*. European Cooperation for Space Standardization, 2009.

[ECS17] ECSS. *ECSS-Q-ST-80C Rev.1 - Software product assurance*. European Cooperation for Space Standardization, 2017.

[Fla23] Paddy Flanagan. Generating Tests for the 'Space Profile' of a Real-Time Operating System (RTEMS) from Promela Models. Master of Engineering (Computer Engineering), School of Engineering, Trinity College, Dublin 2, Ireland, April 2023.

[HBB+09] Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul J. Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy A. Vilkomir, Martin R. Woodward, and Hussein Zedan. Using formal specifications to support testing. *ACM Comput. Surv.*, 41(2):9:1–9:76, 2009.

[Hol04] Gerard J. Holzmann. *The SPIN Model Checker - primer and reference manual*. Addison-Wesley, 2004.

[Hun23] James Gooding Hunt. Software Support for Model Based Test Generation. Master of Engineering (Computer Engineering), School of Engineering, Trinity College, Dublin 2, Ireland, April 2023.

[Jaś22] Jerzy Jaśkuć. SPIN/Promela-Based Test Generation Framework for RTEMS Barrier Manager. Master of science in Computer Science (MCS), School of Computer Science and Statistics, Trinity College, Dublin 2, Ireland, April 2022.

[Jen21] Robert Jennings. Formal Verification of a Real-Time Multithreaded Operating System. Master of science in Computer Science (MCS), School of Computer Science

and Statistics, Trinity College, Dublin 2, Ireland, August 2021.

[Lyn22] Eoin Lynch. Using Promela/SPIN to do Test Generation for RTEMS-SMP. Master of Engineering (Computer Engineering), School of Engineering, Trinity College, Dublin 2, Ireland, April 2022.

[RTEa] RTEMS Classic API Guide. `https://docs.rtems.org/branches/master/c-user.pdf`.

[RTEb] RTEMS Software Engineering. `https://docs.rtems.org/branches/master/eng.pdf`.