Verified Systems International GmbH

Benchmark Analysis of GTL-Backends using Client-Server Mutex*

Document-Identification:	Verified-WHITEPAPER-001-2012 Issue 1.2 Release
Author:	M. Oliver Möller <m@verified.de>

Verified Systems International GmbH
Prof. Dr. Jan Peleska
Dr. Cornelia Zahlten
Am Fallturm 1
D-28359 Bremen
Tel.: +49 421 572040
Fax: +49 421 5720422
e-mail: info@verified.de

Amtsgericht Bremen HR B 18341
Ust-IdNr. DE196934881
Geschäftsführerin Dr. Cornelia Zahlten

(*) This work was developed during the course of the project "Verifikation von Systemen synchroner Softwarekomponenten" (VerSyKo) funded by the German ministry for education and research (BMBF).

Contents

List of Figures

1. Preface

This white-paper reports on benchmark measurements performed for the alternative model-checking backends of the contract specification language GTL.[1]

The GTL has been developed in context of the VerSyKo project[2] — see https://www.tu-braunschweig.de/iti/research/versyko.

The considered backends are the model-checking tools SPIN [2] and UPPAAL [3]. The objective of this document is to compare their performance with respect to time and memory consumption for equivalent inputs.

More details—including the formal definition of the GTL—can be found in [1].

Tool Version Information.
For the GTL tool, version 0.1 (2012-01-09) is used.

For UPPAAL, the 64-bit version 4.1.7 is used.

For SPIN, version 6.1.0 is used.

See Appendix C for more details.

Test Platform Information.
All run-time tests have been executed on a 2.80GHz Intel® Xeon® CPU with 24GB of main memory and 12288 KB of cache. The machine (bull) provides multiple CPUs (24), but the tool make use of only one CPU (with 100% load). See Appendix D for more details.

Time and memory data has been retrieved with the system utility

```
/usr/bin/time
```

`time` version: GNU time 1.7. See Appendix A.2 for more details.

1.1 Changes with Respect to Previous Releases

Issue	Changes
1.2	Supplemented BMBF Reference on title page.
1.1	Considered compile-time optimization options `-O2`, `-O3` for SPIN. Removed some redundant UPPAAL plots (using `-C` or `-Z`).
1.0	(Initial Release)

[1] GTL: GALS translation language, where GALS stands for *globally asynchronous, locally synchronous*.
[2] Verifikation von Systemen synchroner Software-Komponenten

2. The Benchmark Example: Client-Server Mutex

SPIN is a LTL model checker, while UPPAAL treats a (small) subset of timed CTL (TCTL). Therefore the tools are incomparable in general. However, the common subset of these concepts includes *safety conditions* (also known as "*never claims*"). We will use this subset for our analysis.

2.1 Criteria for Selecting the Example

When selecting a benchmark example, the following criteria should be fulfilled.

(A) *The example should be easy to comprehend.*

Rationale: It should be possible for a human inspector to determine whether a given input represents a valid instance of the example. Otherwise, the generated input data could not be trusted.

(B) *The example should be scalable with respect to size, i.e. provide parametrized instances with increasing state space.*

Rationale: For model-checking, it is common that "small" inputs can be processed without meaningful time/memory consumption and "large" inputs yield out-of-memory conditions. To allow for a meaningful comparison, we would like to gradually increase the input size (i.e. size of the state space), until we reach the limits of feasibility or user patience.

(C) *The example should provide a non-trivial safety property (that holds).*

Rationale: A meaningful comparison of tools requires comparable tasks they perform. Model-checking a safety-condition (successfully) corresponds to a statement with respect to the complete (reachable) state space; since the input data is equivalent, so is their state space.

2.2 Client-Server Mutex Protocol

We found an example that meets all the above criteria with a Client-Server Mutex protocol.

In general, Mutex (or *mutual exclusion*) is a property of parallel processes that compete for a shared resource. The resource can be allocated to at most one process, thus a protocol has to be established that (I) allows in principle every process to get the resource and (II) prevents two (or more) processes from using the resource at the same time.

A process that got hold of the resource is said to be in the *critical section*. (I) is a *fairness condition* ("something good will eventually happen to everyone") on the protocol, while (II) is a *safety condition* ("nothing bad will happen").

There are various mutex protocols established in the literature (see, e.g.,[4]). The *Client-Server Mutex* is a variation, where N clients may request access to the critical section from a single server. A client only enters the critical section, if the request is granted. After leaving the critical section, the client notifies the server.

The server

- keeps track of who got granted access to the critical section
- and
  - (i) if a client is in the critical section, nobody else is granted access
  - (ii) otherwise, the server grants access to exactly one client (selected from the list of applicants for the critical section)

This protocol is simple (*criterion (A)*) and allows for an arbitrary (fixed) number of clients (*criterion (B)*).

Moreover, the safety condition "at most one client in the critical section" is a purely combinatorial property, i.e., no clock is involved. Thus it can be expressed equivalently both in SPIN and in UPPAAL. Since the property is true (by inspection), the complete state space has to be analyzed by the corresponding tool (*criterion (C)*).

3. Generation of Input Data

This chapter explains how the input data for the model-checking tools have been generated. First a GTL formulation for sequence of clients N has been generated. This has been transformed by the GTL utility to the equivalent SPIN or UPPAAL representation.

3.1 Generation of GTL formulation

The Client-Server Mutex sketched in Section 2.2 can be formulated very systematically in GTL.

The generation here has been done via an AWK-Script (Appendix A.1). The script takes the number N of clients as command line input and prints the GTL-Syntax to `<stdout>`.

Sample output for $N = 2$ and $N = 3$ are included in the Appendix (B.1.1, B.1.2). Proper operation has been validated by (textually) comparing the output of $N = 3$ against the manual formulation, as it has been developed by Technical University Braunschweig in context of the VerSyKo project, see [1].

3.2 Generation of SPIN/PROMELA input data

The input language for the SPIN tool is PROMELA, see [2].

The PROMELA-files were generated by the GTL tool via processing of `mutex_<N>.glt`

```
gtl  mutex_<N>.gtl
```

This yields a `mutex_<N>.pr` file, which contains a LTL-formulation of the safety condition. The output for $N = 2, 3$ is listed in Appendix B.2.1, B.2.2.

Note. The above call to GTL-0.1 (2012-01-09) does not only generate the `*.pr` file, but also processes it with SPIN and runs the corresponding `verifier`. For the measurements, the `verifier` were aborted and re-compiled on the measurement machine (based on the corresponding `*.pr` file).

3.3 Generation of UPPAAL input data

The UPPAAL tool operates on `*.xml` files (following a UPPAAL specific document-type definition), see [3].

The `*.xml` files were generated by the GTL tool via processing of `mutex_<N>.glt`:

```
gtl  -m uppaal  mutex_<N>.gtl
```

This yields a `mutex_<N>.xml` file. The output for $N = 2, 3$ is listed in Appendix B.3.1, B.3.2.

UPPAAL expects model-checking queries (here: the safety condition) to be stored in a separate *query* file, `*.q`.

In version 0.1 (2012-01-09), the GTL tool is not capable of generating this file automatically, since in general not every LTL formula *can* be expressed in the UPPAAL query language.[1]

Therefore, the query files `mutex_<N>.q` have been generated together with the `*.gtl` files by the AWK-Script (Appendix A.1). The output for $N = 2, 3$ is listed in Appendix B.3.1, B.3.2.

[1] A subset of TCTL.

4. The Measurement Process

SPIN-6.1.0 and UPPAAL-4.1.7 have been installed on a high-performance machine (`bull`), which provides 2.80GHz Intel® Xeon® CPU with 24GB of main memory and 12288 KB of cache. The operating system is Linux CentOS release 5.7. More details can be found in Appendix D.

The machine has *not* been reserved exclusively for the benchmark testing; however, there were no processes with substantial memory consumption present during the measurement. The time measurement refers to user time, i.e. the total number of CPU-seconds that the process spent in user mode.[1] The memory measured is the *optimistic* allocation that the Linux kernel allows, i.e. the memory the process *might* make use of at the point of highest memory load.[2]

The UPPAAL tool can directly operate on the `*.xml` and `*.q` files. The command line invocation looks like this:

```
verifyta   mutex_<N>.xml mutex_<N>.q
```

SPIN/PROMELA, the `*.pr` files were transformed to a `verifier` executable, before the measurement was started; this happened in time well below a second and can be neglected. The command line invocation then looks like this:

```
./mutex_<N>-verifier
```

The time and memory information have been recorded by means of the script `measure.bash`, which makes use of the system utility `/usr/bin/time` (see Appendix A.2).

All verification processes were run in sequence (to exclude interaction). Every verification process recorded its options and results (time, memory, verification outcome) to a file. The graphical representation has been derived from these files by means of of the `gnuplot`, see the corresponding `Makefile` in Appendix A.3.

4.1 Selection of Compile-Time / Run-Time Options

The tools SPIN and UPPAAL allow for various user options to modify (and hopefully speed-up) the verification of a given model, see Appendix C.2, C.3. SPIN also allows *compile-time options*, since the verifier is generated by compiling C-file `pan.c` (here: with the gcc).

[1] Comparison of user time/real time shows, that effectively one CPU exclusively executed the model-checking process; in presence of 24 CPUs, this is not surprising.

[2] This explains why some numbers exceed the available 24GB of main memory without using swap.

For UPPAAL, there are no compile-time options available (to the user); it should be noted, however, that a 64-bit executable of UPPAAL is used, since this allows addressing of more than 4GB of memory.

The following selection of (combinations of) tool options have been made.

SPIN option combinations used:

Compile Time	Run Time
	`-a`
	`-m999k`
	`-m9999999k -a`
`-DSAFETY`	`-m9999999k`
`-O2`	`-m9999999k -a`
`-O2 -DSAFETY`	`-m9999999k`
`-O3`	`-m9999999k -a`
`-O3 -DSAFETY`	`-m9999999k`

UPPAAL option combinations used:

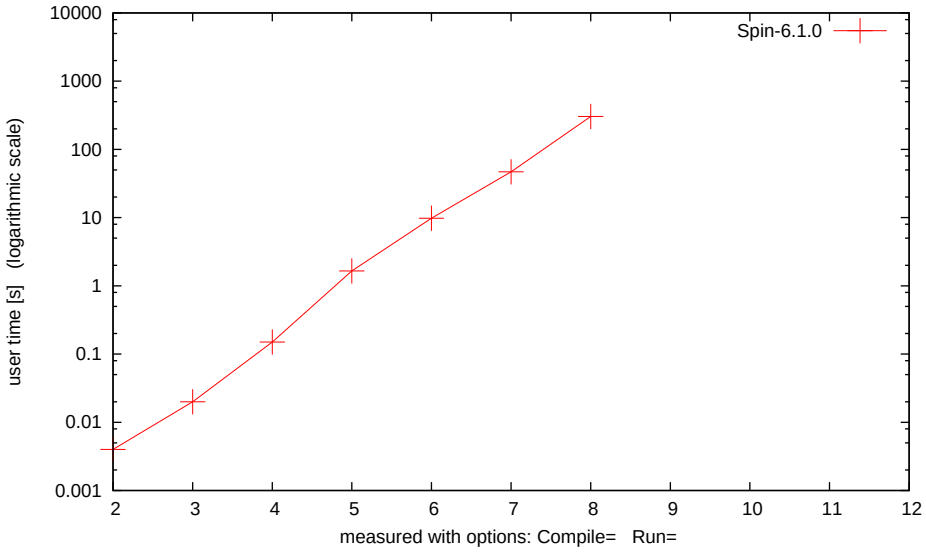
Compile Time	Run Time
(N/A)	`-S 0`
(N/A)	`-S 1`
(N/A)	`-S 2`
(N/A)	`-S 2 -A`
(N/A)	`-S 0 -C`
(N/A)	`-S 1 -C`
(N/A)	`-S 2 -C`
(N/A)	`-S 2 -Z`
(N/A)	`-S 2 -n 0`
(N/A)	`-S 2 -n 1`
(N/A)	`-S 2 -n 2`
(N/A)	`-S 2 -n 3`
(N/A)	`-S 2 -n 4`

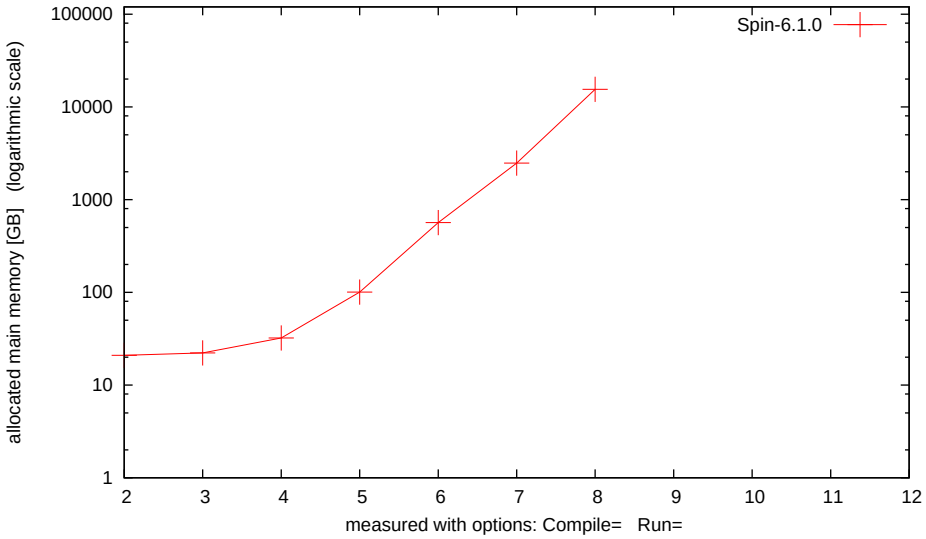
4.2 Graphical Presentation of Results

In order to visualize the slope of time/memory consumption for increasing N, the data has been plotted with a logarithmic-scaled y-axis.

The axes were set to the same range for all plots to allow for visual comparison.

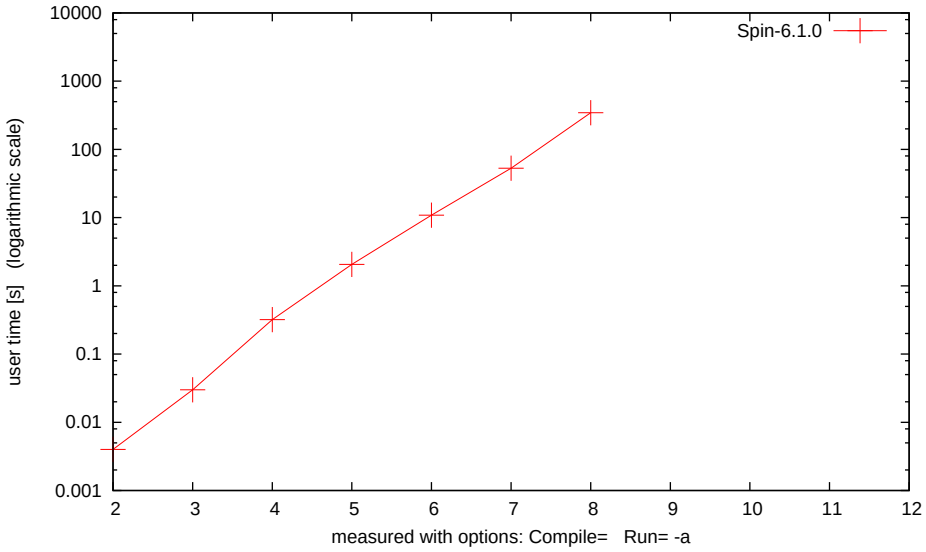
4.2.1 SPIN measurement with Compile=(none) Run=(none)

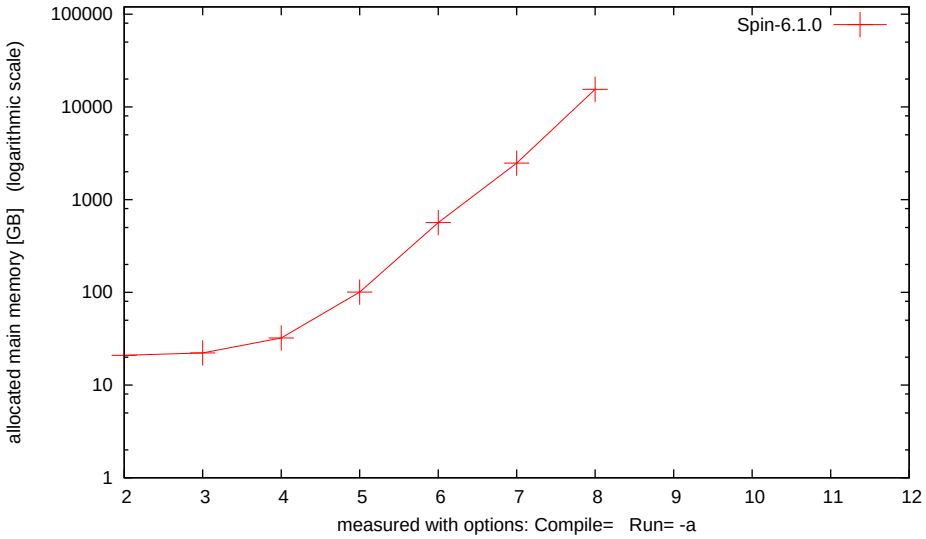




Note: The SPIN measurement is *unreliable* (i.e., incomplete) here due to command line options, see comments on page 35.

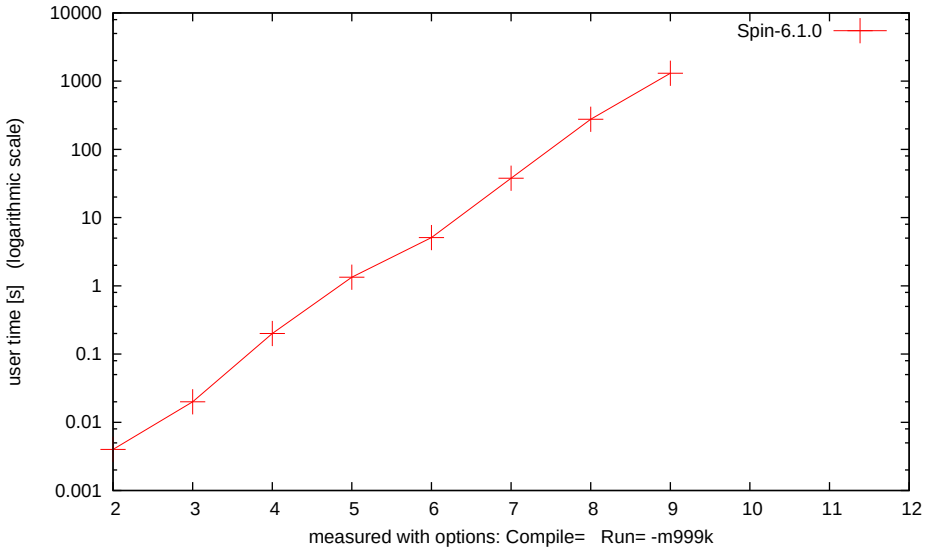
4.2.2 SPIN measurement with Compile=(none) Run=`-a`

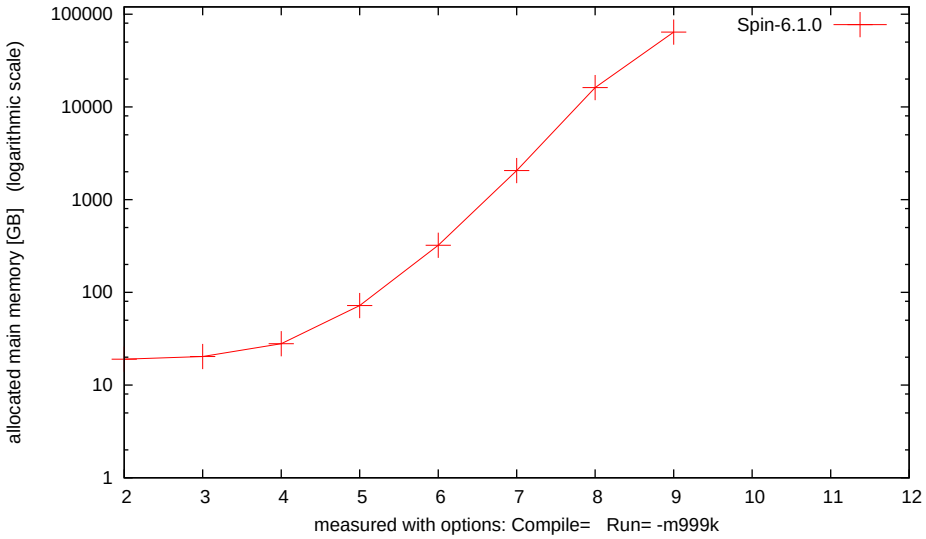




Note: The SPIN measurement is *unreliable* (i.e., incomplete) here due to command line options, see comments on page 35.

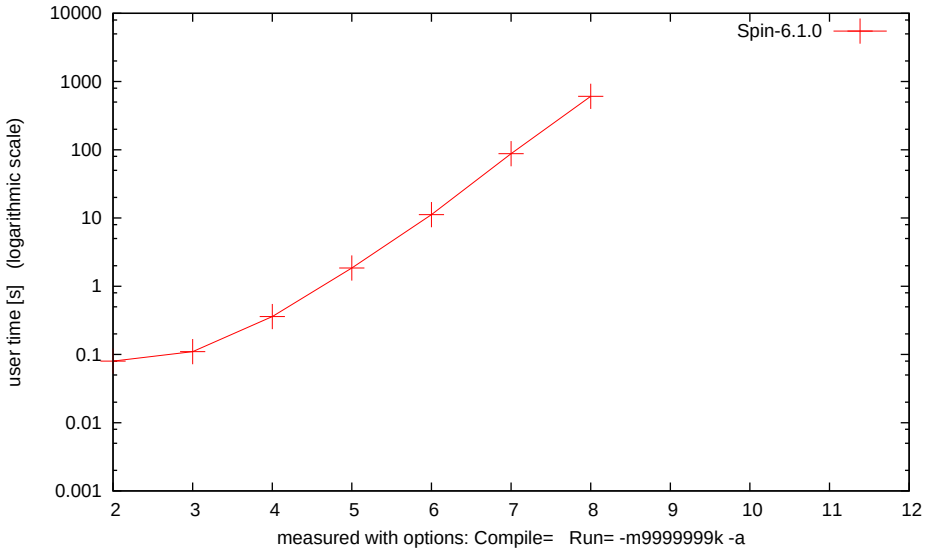
4.2.3 SPIN measurement with Compile=(none) Run=`-m999k`

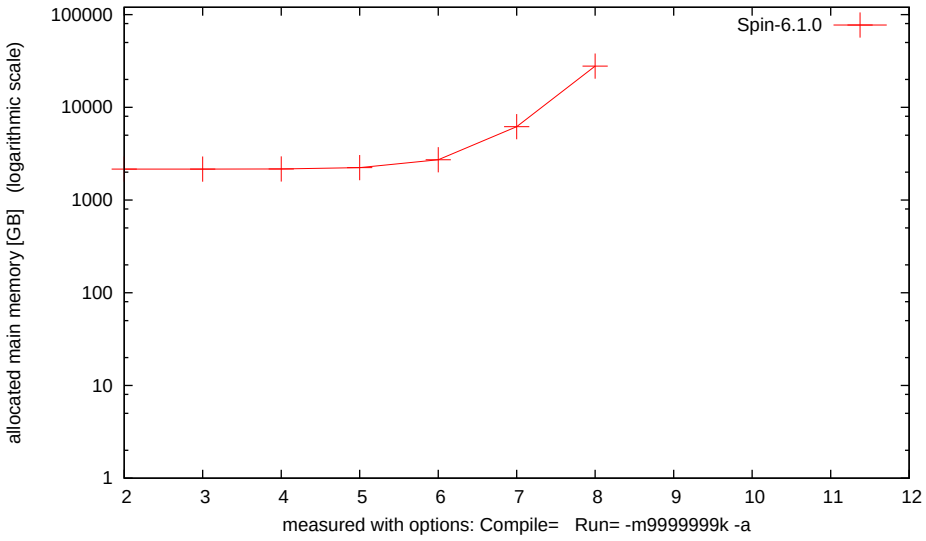




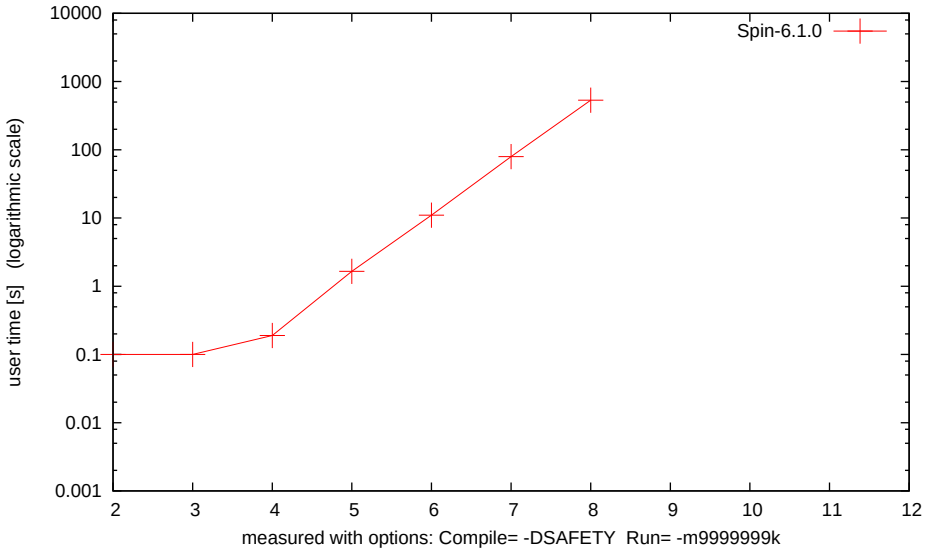
Note: The SPIN measurement is *unreliable* (i.e., incomplete) here due to command line options, see comments on page 35.

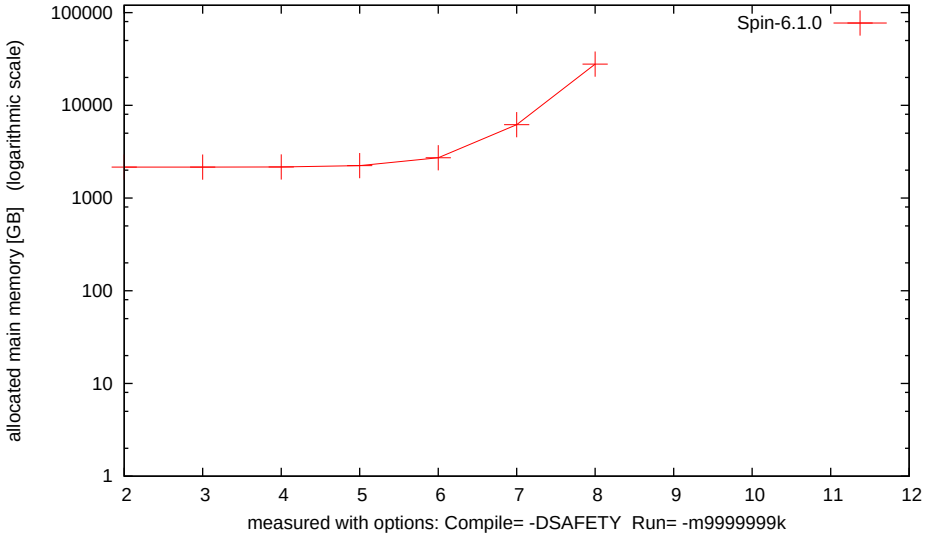
4.2.4 SPIN measurement with Compile=(none) Run=`-m9999999k -a`



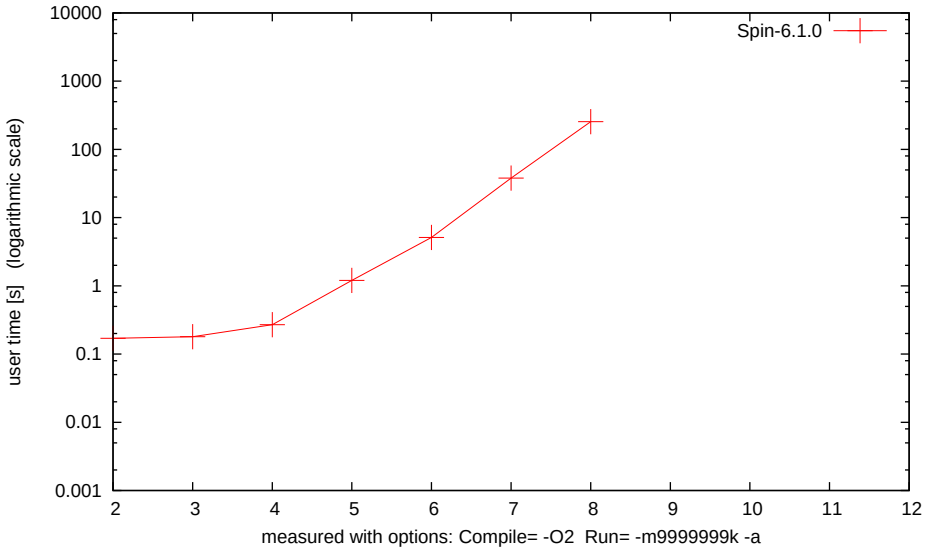


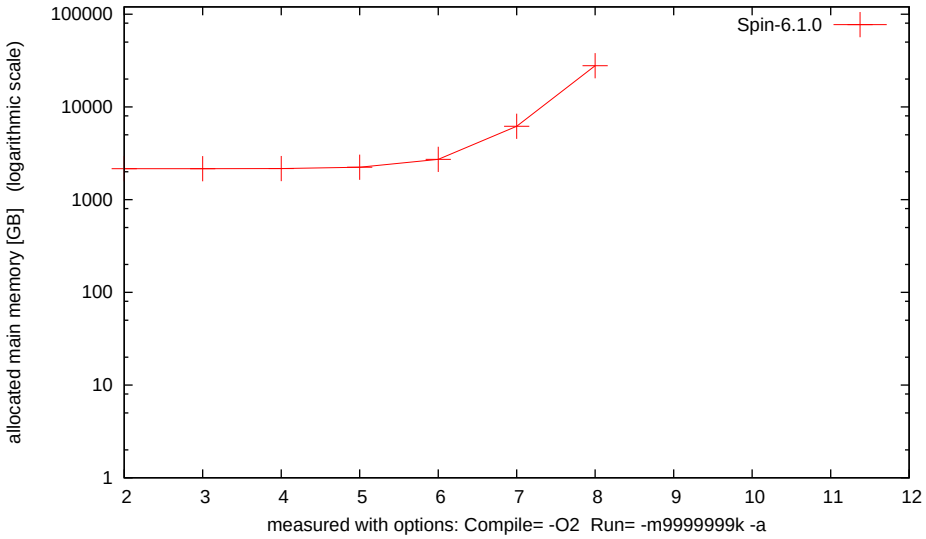
4.2.5 SPIN measurement with Compile=`-DSAFETY` Run=`-m9999999k`



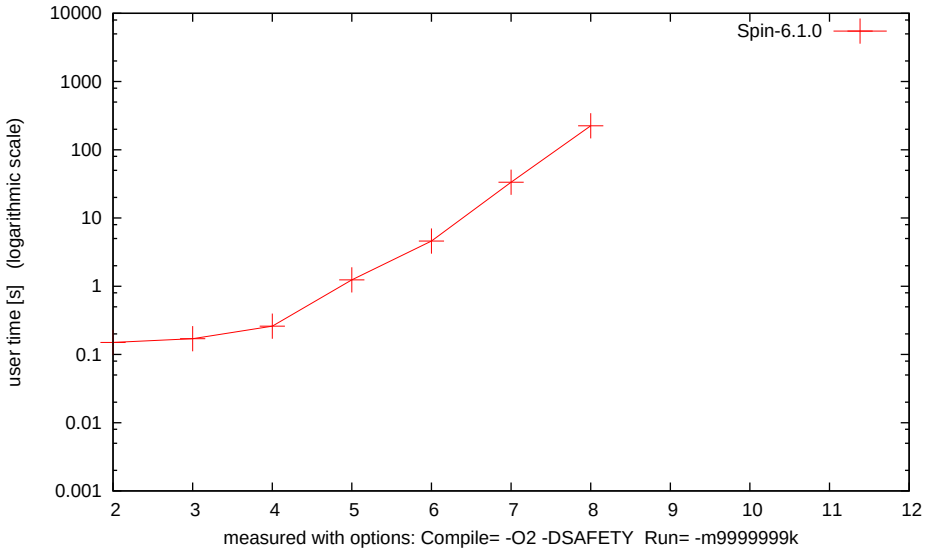


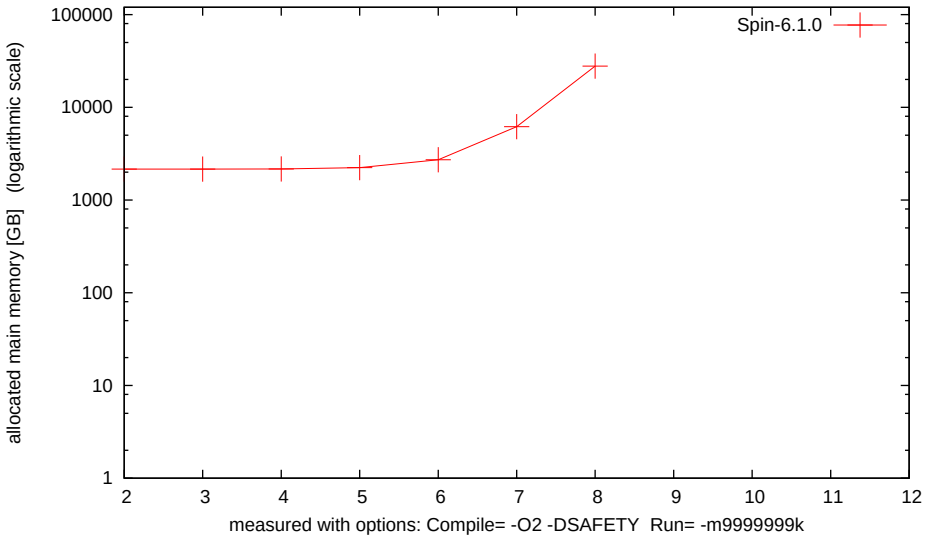
4.2.6 SPIN measurement with Compile=`-O2` Run=`-m9999999k -a`



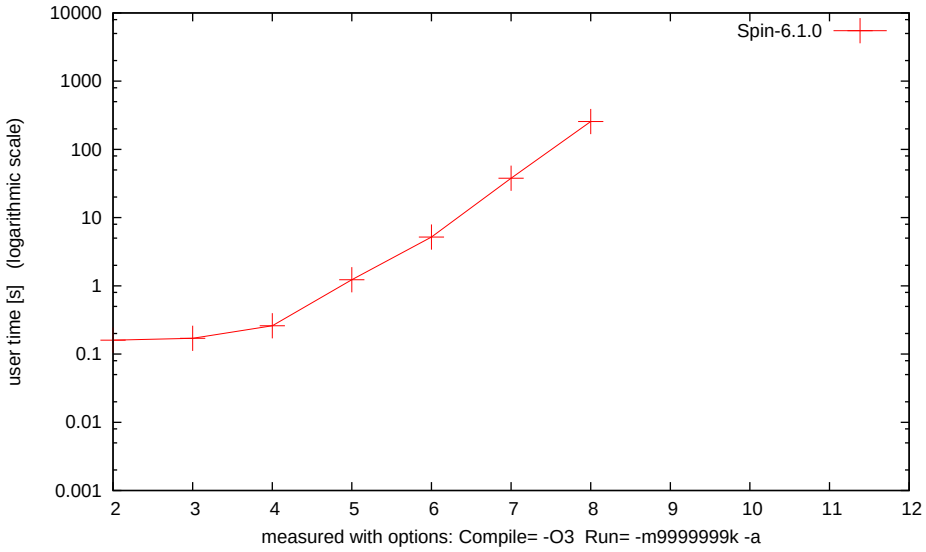


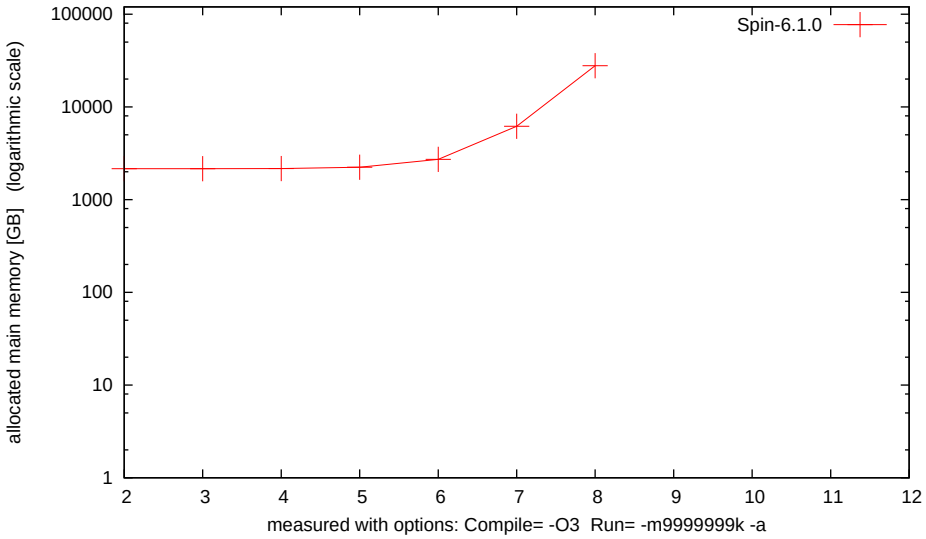
4.2.7 SPIN measurement with Compile=`-O2 -DSAFETY` Run=`-m9999999k`



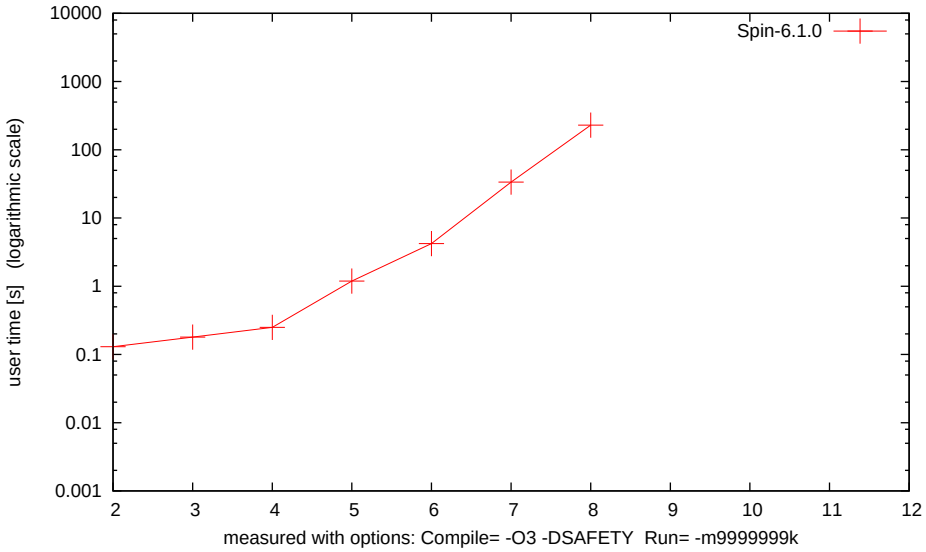


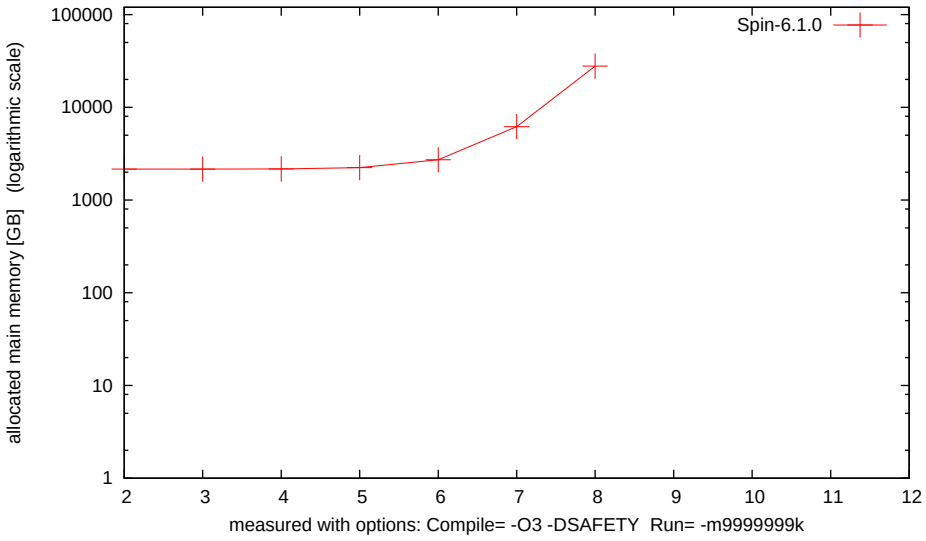
4.2.8 SPIN measurement with Compile=`-O3` Run=`-m9999999k -a`



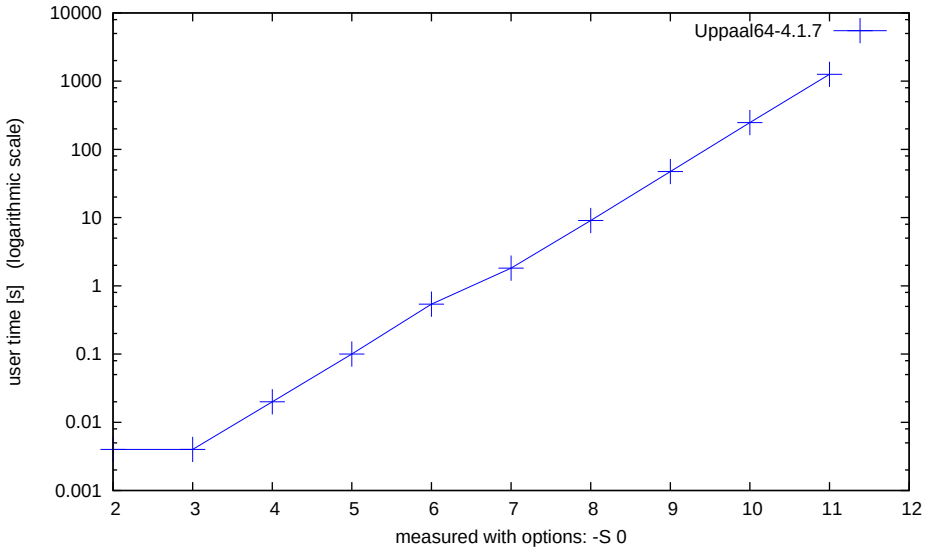


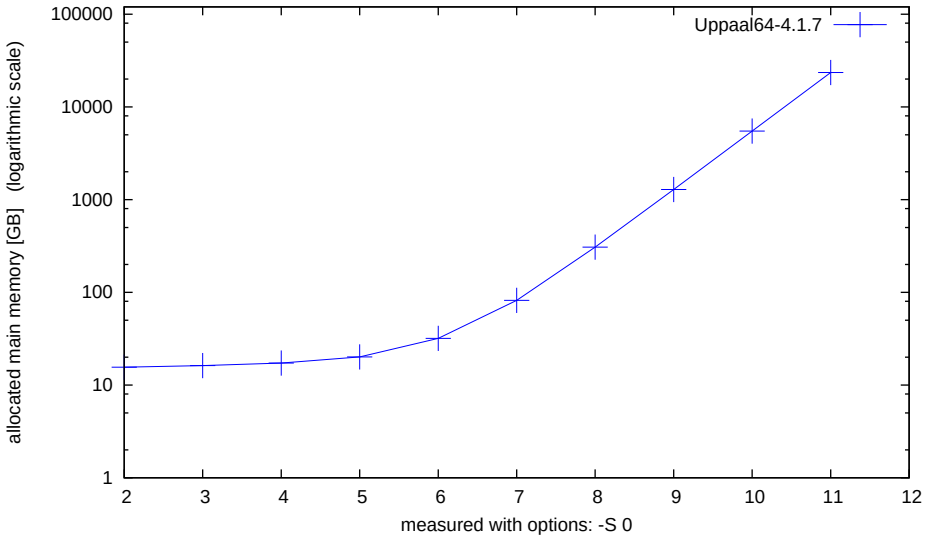
4.2.9 SPIN measurement with Compile=`-O3 -DSAFETY` Run=`-m9999999k`



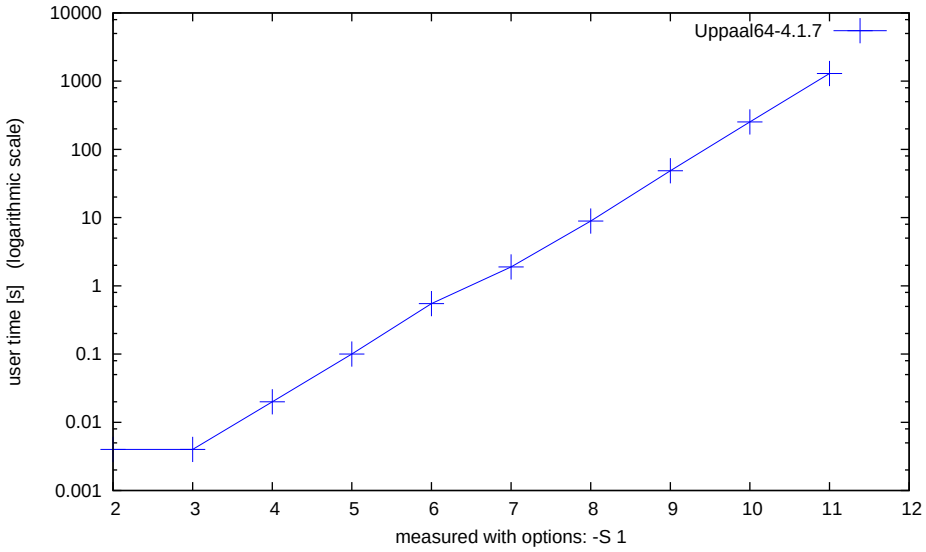


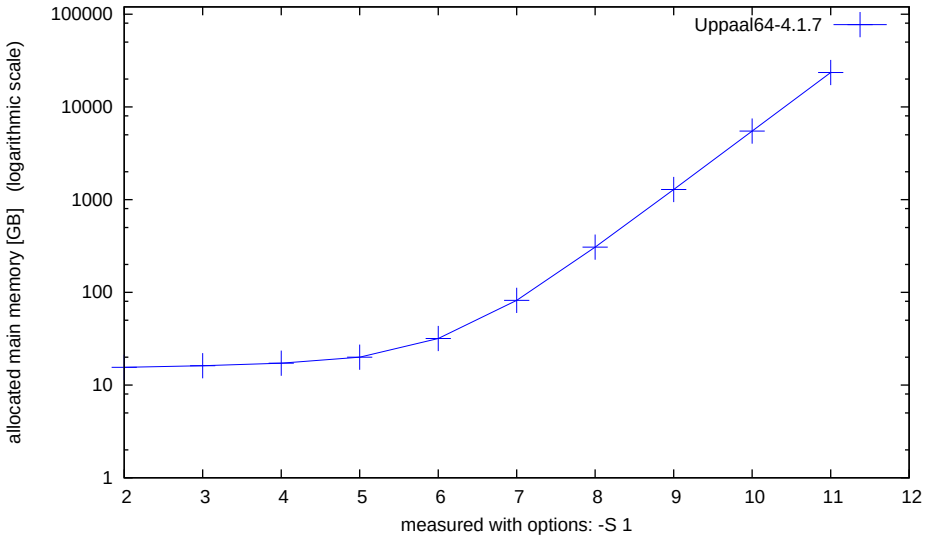
4.2.10 UPPAAL measurement with options `-S 0`



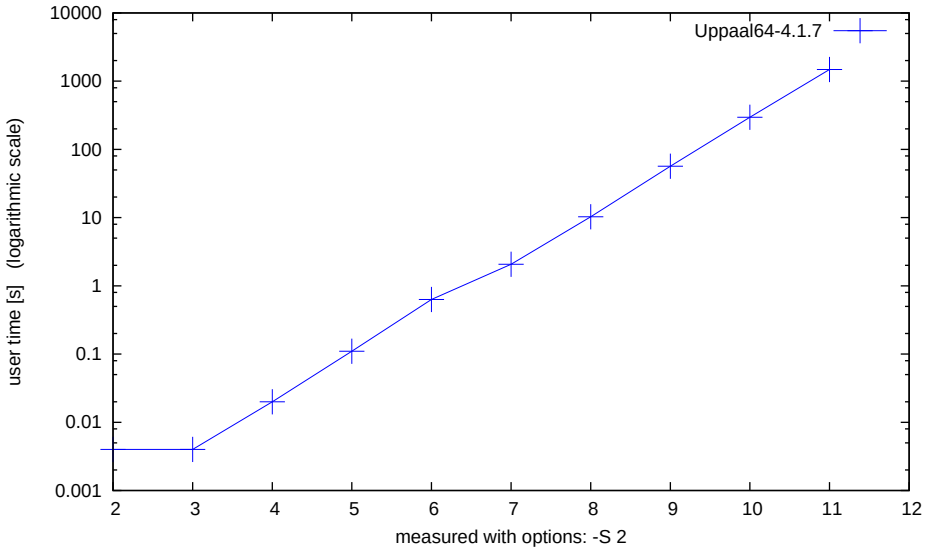


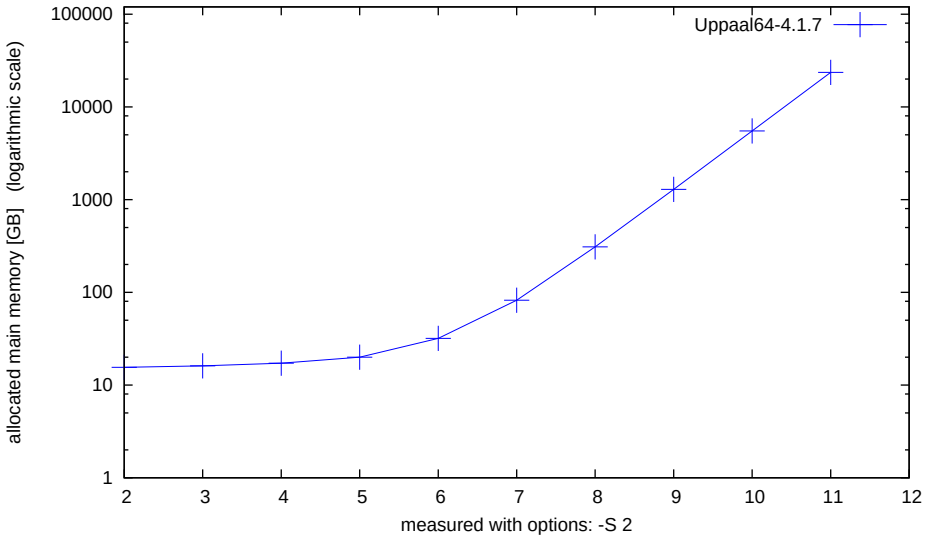
4.2.11 UPPAAL measurement with options `-S 1`



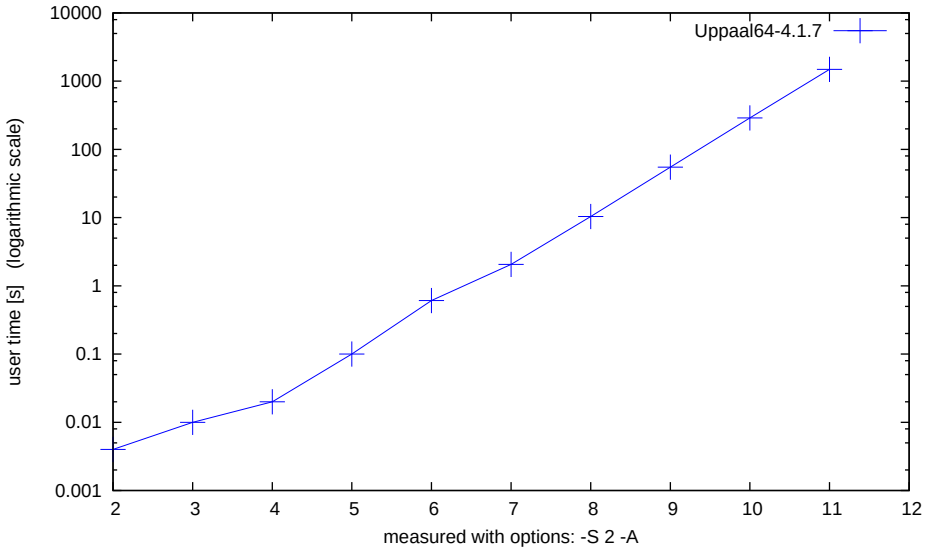


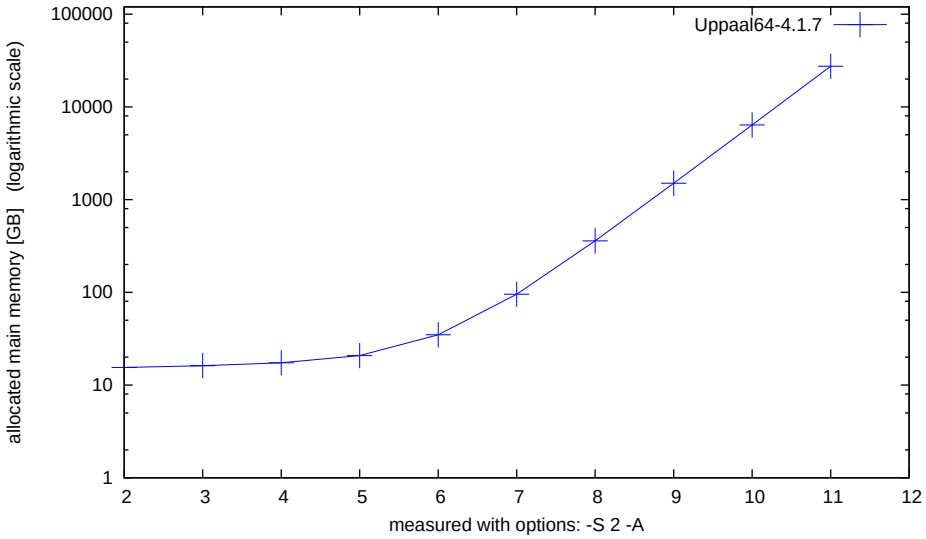
4.2.12 UPPAAL measurement with options `-S 2`



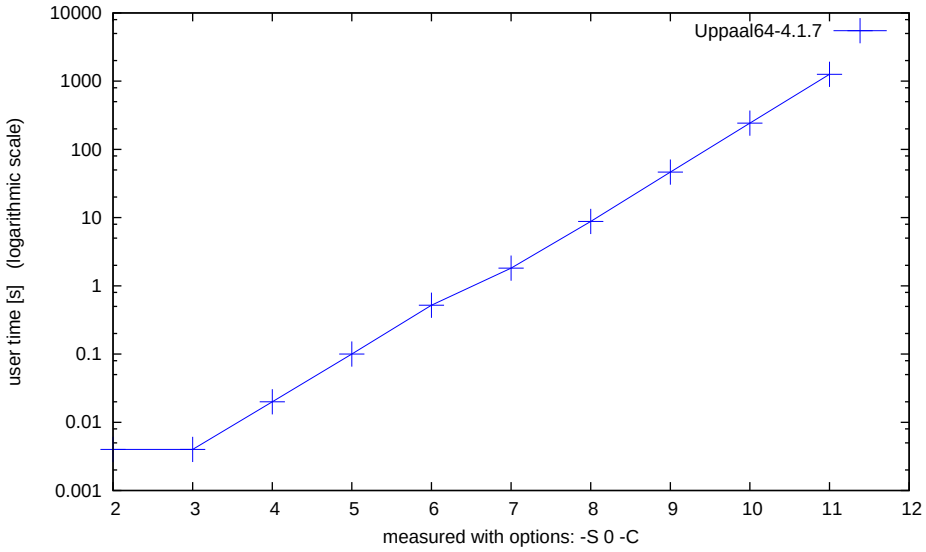


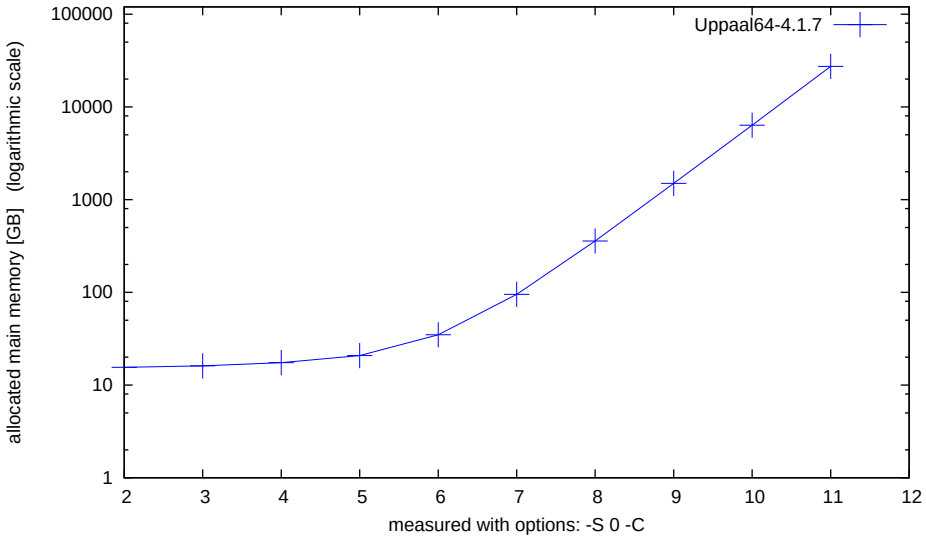
4.2.13 UPPAAL measurement with options `-S 2 -A`



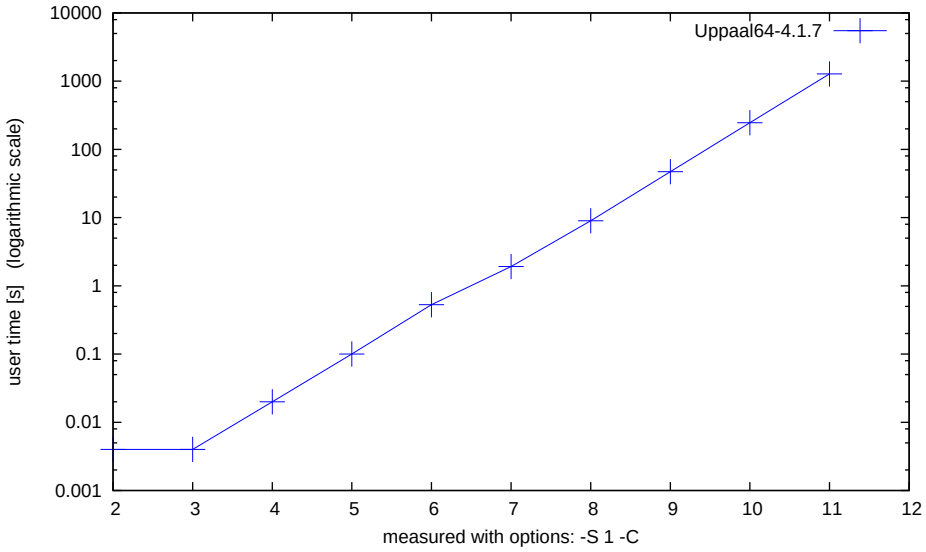


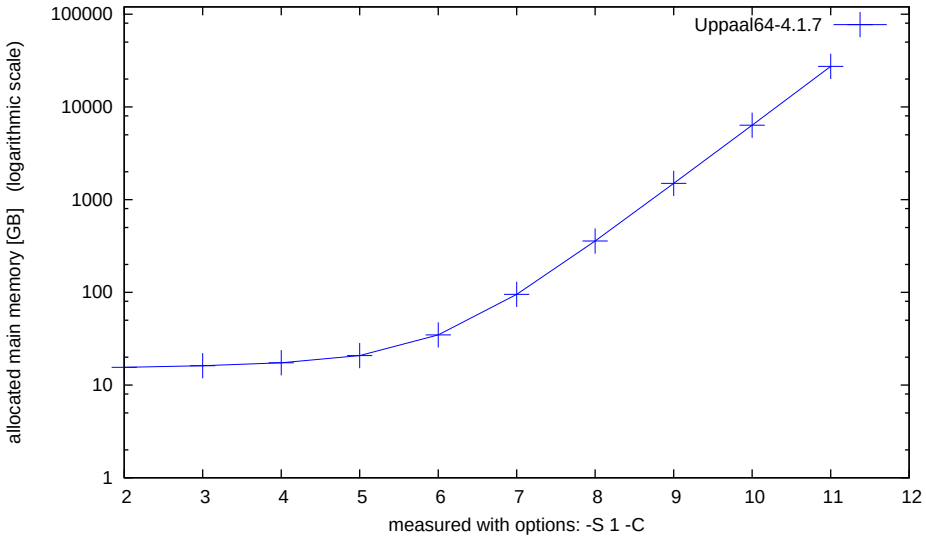
4.2.14 UPPAAL measurement with options `-S 0 -C`



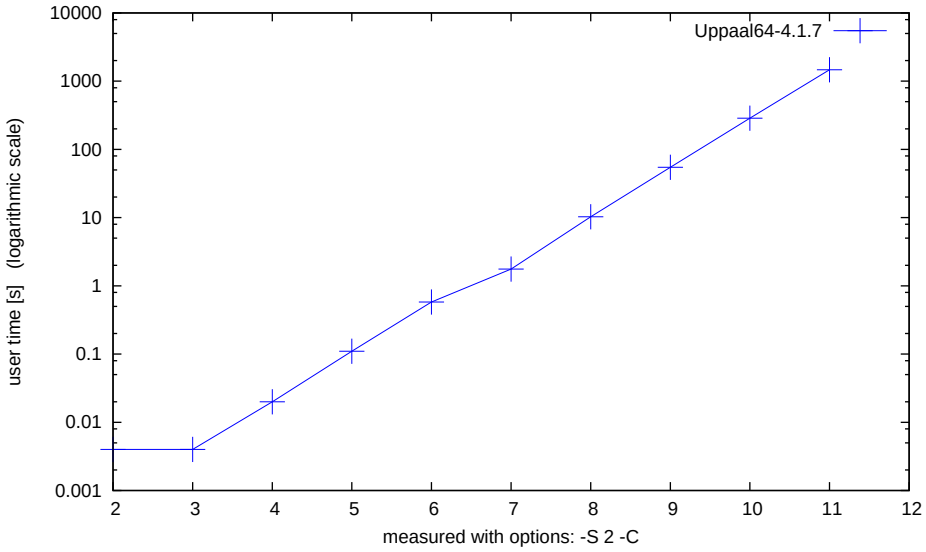


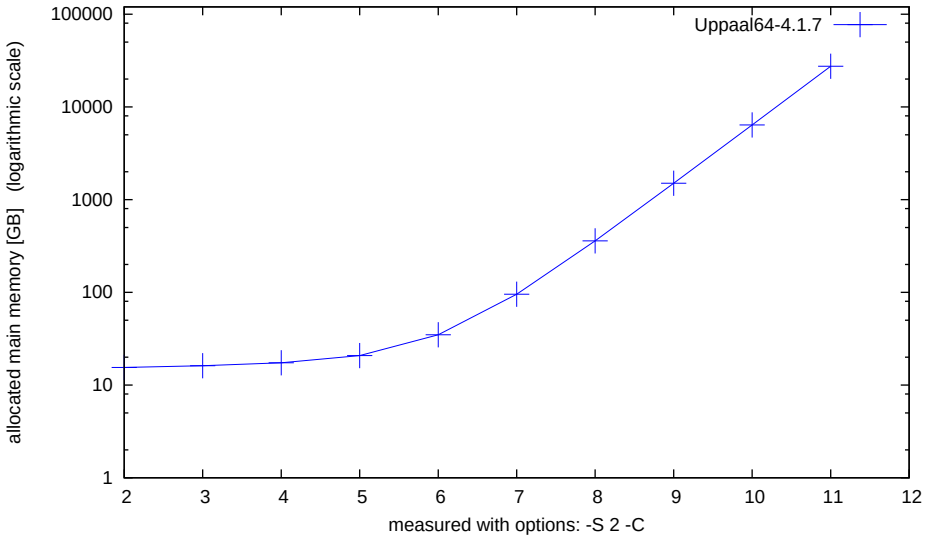
4.2.15 UPPAAL measurement with options `-S 1 -C`



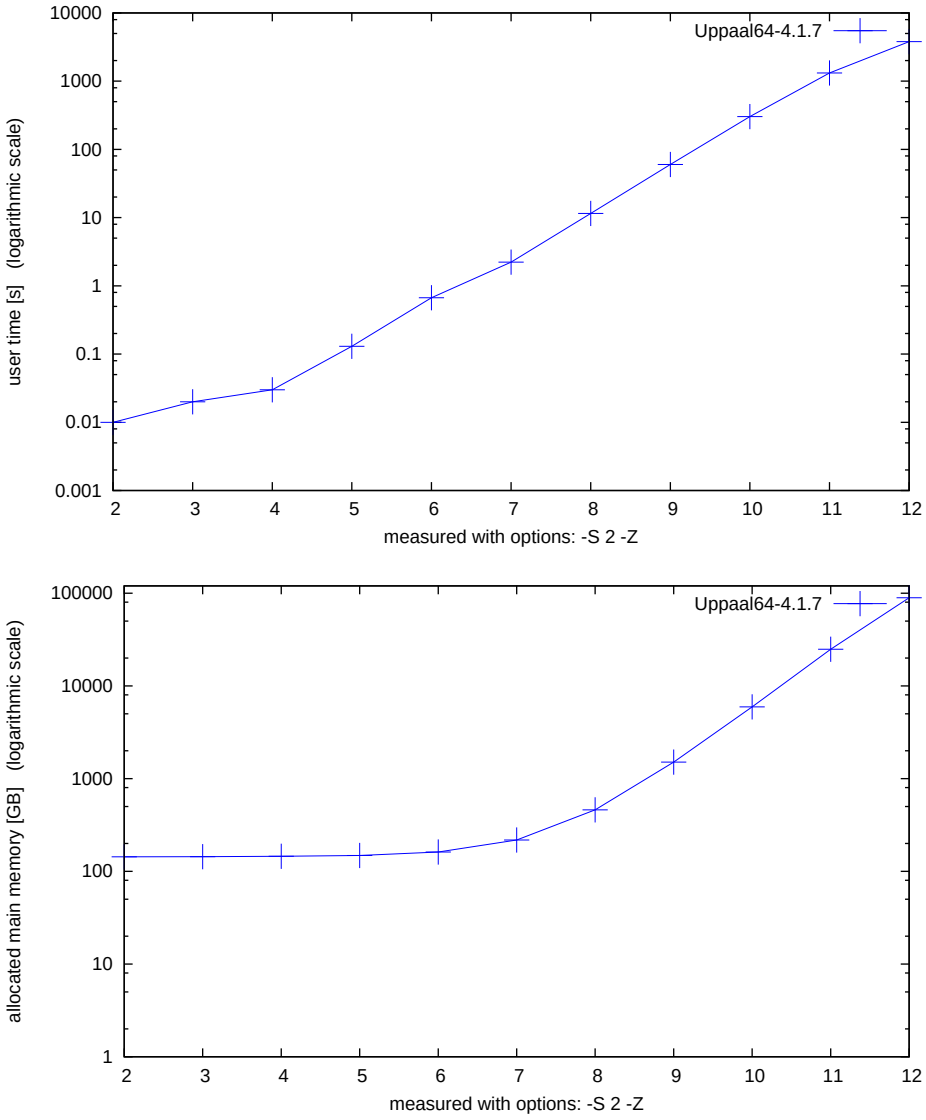


4.2.16 UPPAAL measurement with options `-S 2 -C`



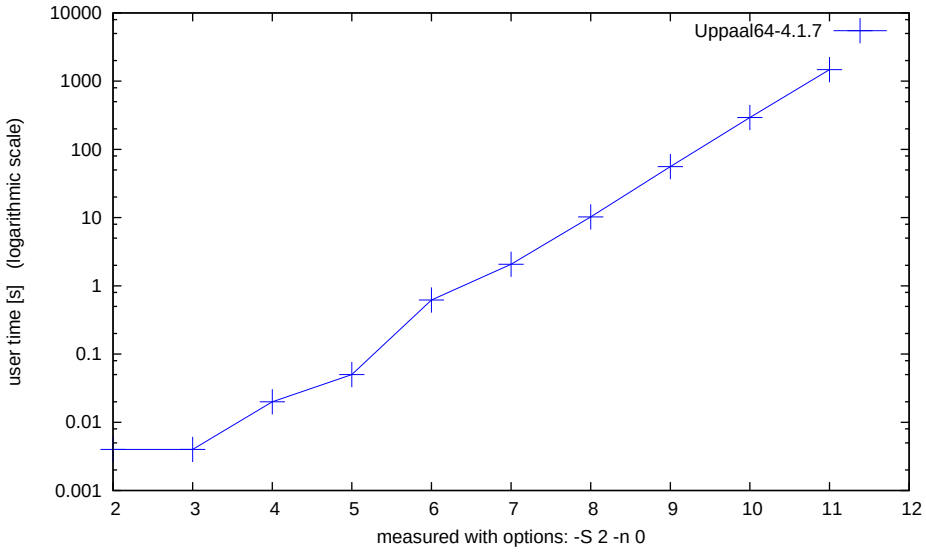


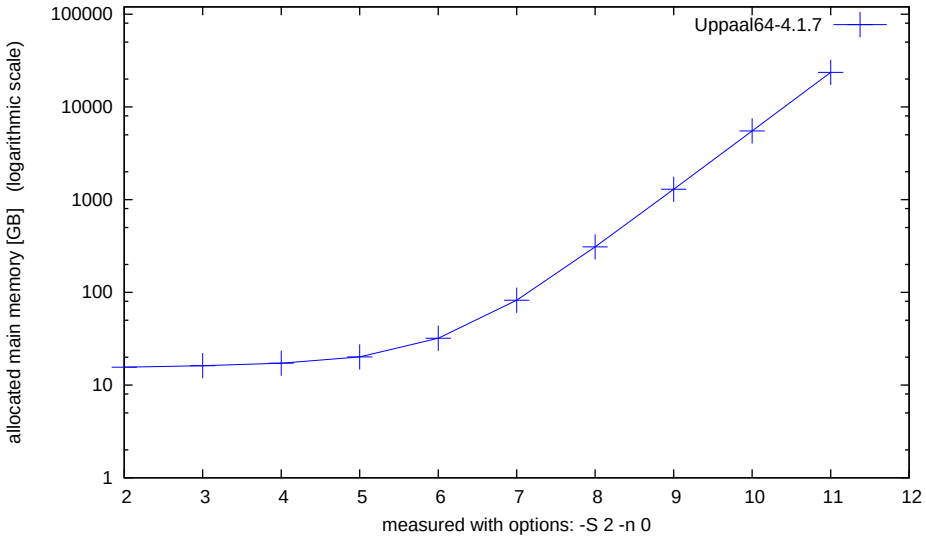
4.2.17 UPPAAL measurement with options `-S 2 -Z`



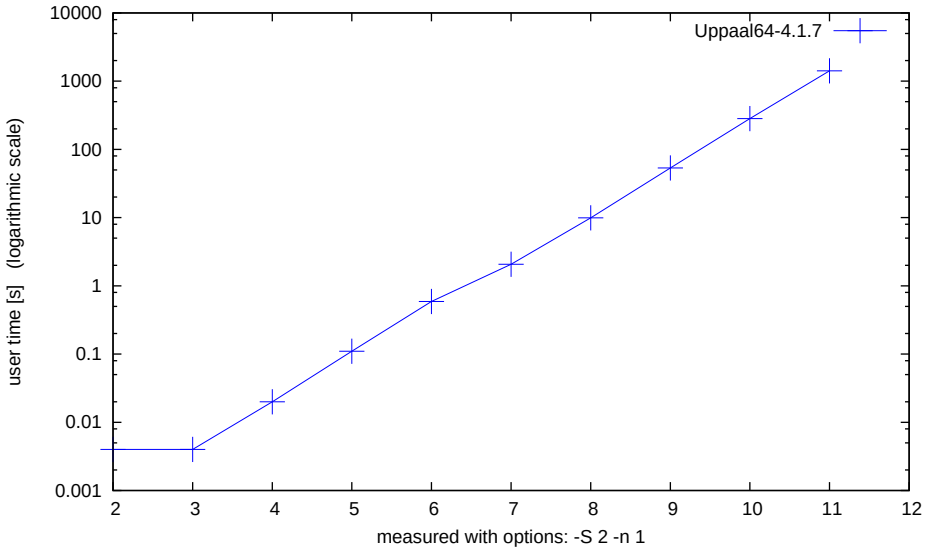
Note: The UPPAAL measurement is *unreliable* (i.e., incomplete) here due to command line options, see comments on page 35.

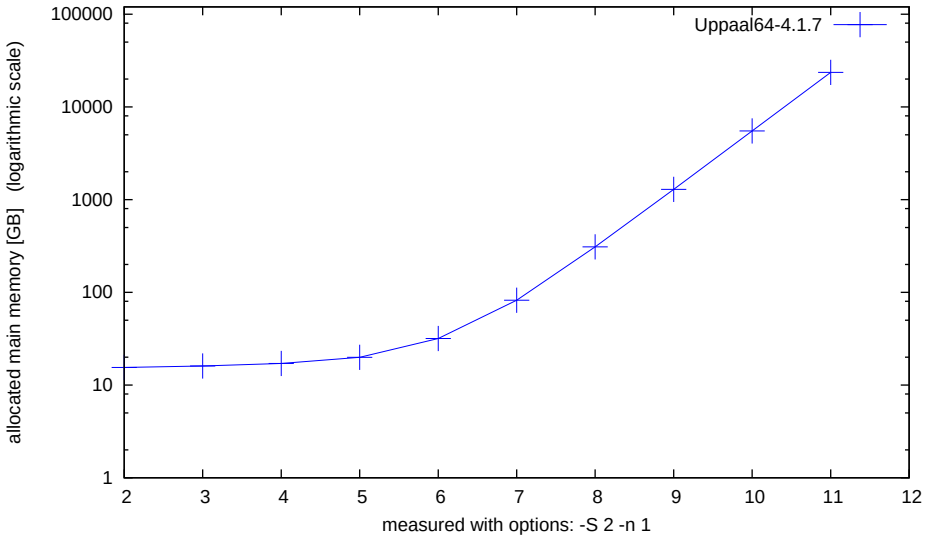
4.2.18 UPPAAL measurement with options `-S 2 -n 0`



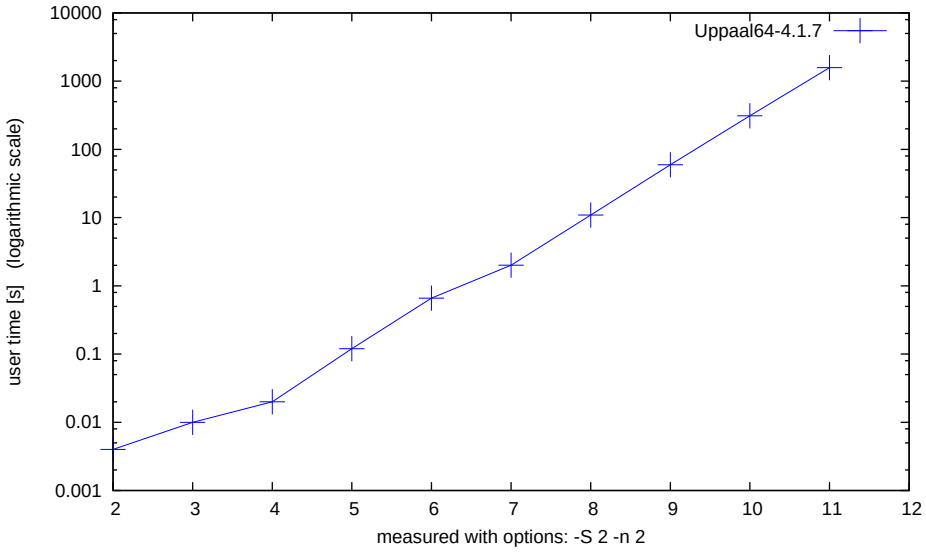


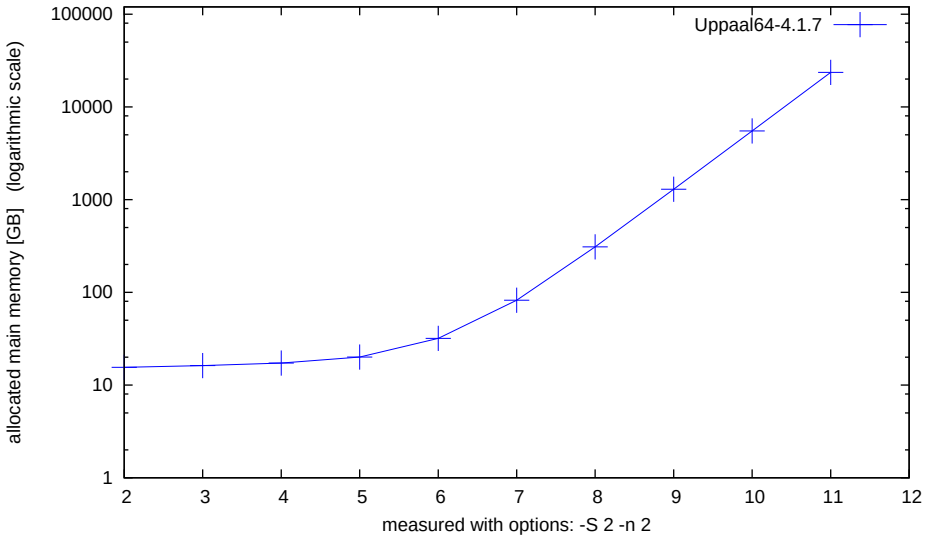
4.2.19 UPPAAL measurement with options `-S 2 -n 1`



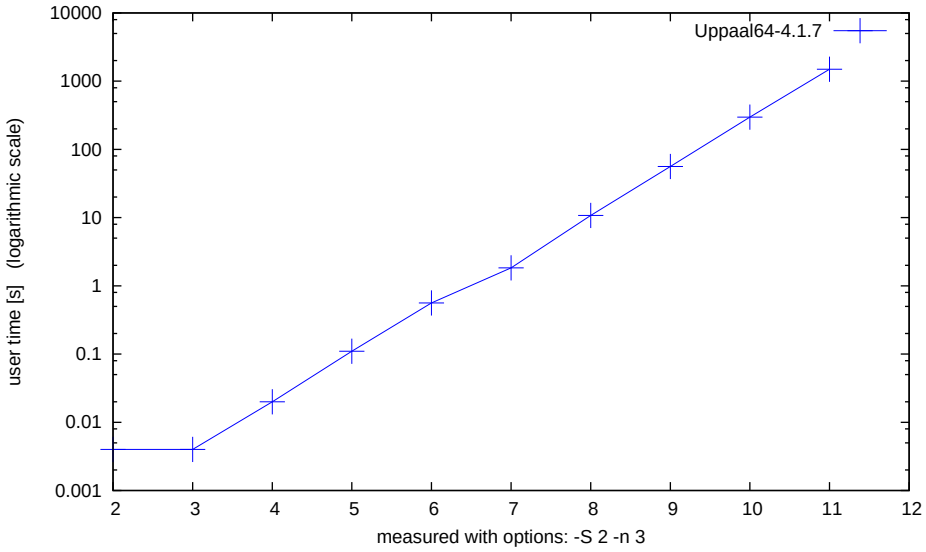


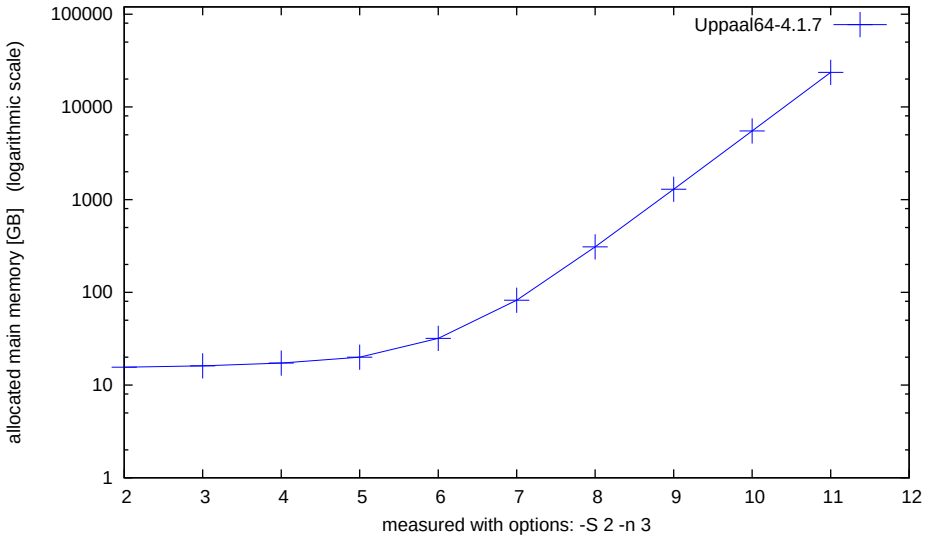
4.2.20 UPPAAL measurement with options `-S 2 -n 2`



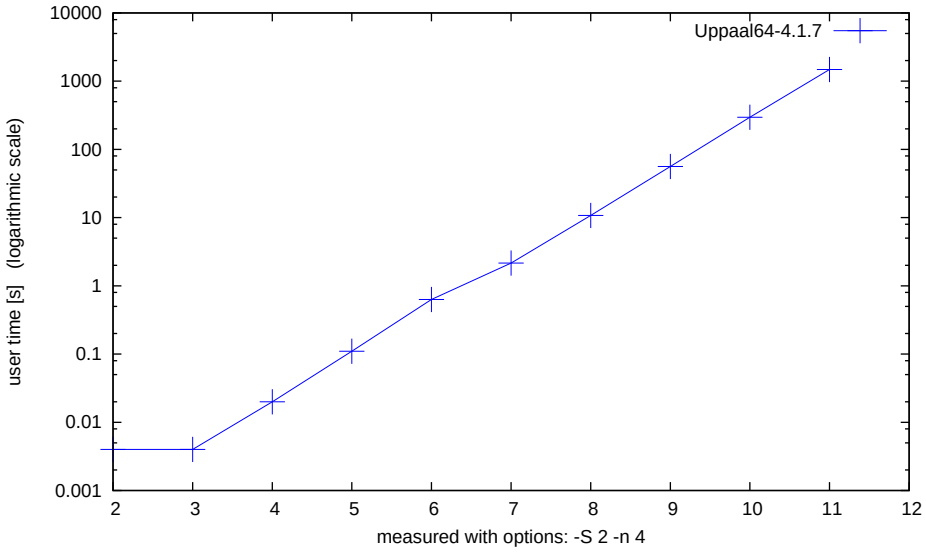


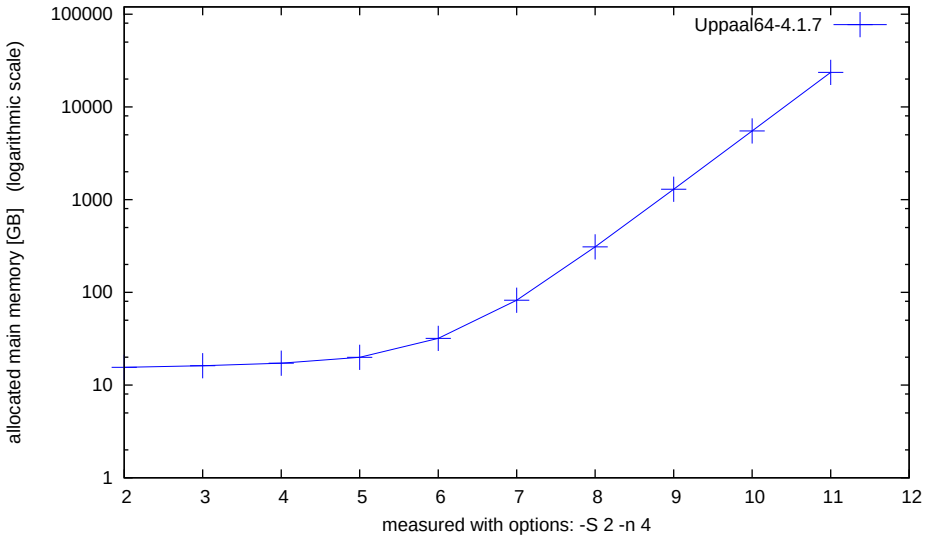
4.2.21 UPPAAL measurement with options `-S 2 -n 3`





4.2.22 UPPAAL measurement with options `-S 2 -n 4`





4.3 Tabular Presentation of Results

This section lists the measurements in a tabular manner. The digits after the second fractional digit (i.e. everything after 0.00) have been truncated to meet width conditions.

Notes on unreliable executions.

(*) Here, the SPIN results are unreliable (i.e., do not give proof), due to option `-m999k`:

The `mutex_*-verifier` output shows the line

```
error:  max search depth too small
```

(‡) Here, the UPPAAL results are unreliable (i.e., do not give proof), due to option `-Z` (bit state hashing):

The `verifyta` output shows the line

```
- Property MAY be satisfied.
```

These are *not* tool defects. For some run-time options the results are inexact in one direction; if a violation would have been found, this would have been reliable.

For a description of the tool options refer to the Appendix (SPIN: C.2; UPPAAL: C.3).

4.3.1 Time Consumption [s]

Compile	Run	2	3	4	5	6	7	8	9	10	11	12
		0.00	0.02	0.15	1.65	9.77	46.82	304.10				(*)
	-a	0.00	0.03	0.32	2.06	10.88	53.07	344.71				(*)
	-m999k	0.00	0.02	0.20	1.34	5.09	37.79	276.30	1307.00			(*)
	-m9999999k -a	0.08	0.11	0.36	1.85	11.22	87.68	606.76				
-DSAFETY	-m9999999k	0.10	0.10	0.19	1.65	11.01	79.59	532.94				
-O2	-m9999999k -a	0.17	0.18	0.27	1.20	5.11	38.04	255.48				
-O2 -DSAFETY	-m9999999k	0.15	0.17	0.26	1.24	4.60	33.45	225.12				
-O3	-m9999999k -a	0.16	0.17	0.26	1.23	5.20	37.83	256.15				
-O3 -DSAFETY	-m9999999k	0.13	0.18	0.25	1.19	4.22	33.68	229.37				

Figure 4.1: Time Consumption [s] of SPIN with Various Compile-time and Run-time Options.

Run	2	3	4	5	6	7	8	9	10	11	12
-S 0	0.00	0.00	0.02	0.10	0.54	1.82	9.08	47.43	247.09	1261.18	
-S 1	0.00	0.00	0.02	0.10	0.55	1.89	8.90	48.59	252.34	1296.53	
-S 2	0.00	0.00	0.02	0.11	0.63	2.07	10.31	56.68	295.80	1483.89	
-S 2 -A	0.00	0.01	0.02	0.10	0.61	2.06	10.37	54.98	289.80	1490.53	
-S 0 -C	0.00	0.00	0.02	0.10	0.52	1.82	8.77	46.51	242.52	1259.12	
-S 1 -C	0.00	0.00	0.02	0.10	0.53	1.92	8.97	47.16	245.69	1276.53	
-S 2 -C	0.00	0.00	0.02	0.11	0.58	1.76	10.31	54.71	286.98	1470.58	
-S 2 -Z	0.01	0.02	0.03	0.13	0.67	2.23	11.53	60.11	302.63	1316.19	3806.93 (‡)
-S 2 -n 0	0.00	0.00	0.02	0.05	0.62	2.07	10.26	55.93	292.92	1476.42	
-S 2 -n 1	0.00	0.00	0.02	0.11	0.59	2.07	9.93	53.35	283.26	1418.98	
-S 2 -n 2	0.00	0.01	0.02	0.12	0.66	2.01	10.91	59.51	310.42	1577.45	
-S 2 -n 3	0.00	0.00	0.02	0.11	0.56	1.83	10.74	56.23	296.44	1492.76	
-S 2 -n 4	0.00	0.00	0.02	0.11	0.63	2.16	10.75	56.04	295.61	1485.28	

Figure 4.2: Time Consumption [s] of UPPAAL with Various Run-time Options.

4.3.2 Memory Consumption [MB]

Compile	Run	2	3	4	5	6	7	8	9	10	11	12
		20.90	22.25	32.18	100.62	566.45	2482.18	15477.00				(*)
	`-a`	20.90	22.25	32.18	100.62	566.46	2482.18	15476.98				(*)
	`-m999k`	19.01	20.32	27.90	71.93	321.59	2061.00	16182.60	64105.17			(*)
	`-m9999999k -a`	2155.03	2156.37	2166.32	2234.75	2718.60	6189.29	27875.21				
`-DSAFETY`	`-m9999999k`	2155.01	2156.34	2165.48	2234.65	2718.57	6188.85	27875.20				
`-O2`	`-m9999999k -a`	2154.93	2156.25	2166.17	2234.57	2718.42	6189.06	27874.93				
`-O2 -DSAFETY`	`-m9999999k`	2154.92	2156.23	2165.37	2234.50	2718.39	6188.62	27874.92				
`-O3`	`-m9999999k -a`	2155.01	2156.34	2166.28	2234.70	2718.56	6189.20	27875.10				
`-O3 -DSAFETY`	`-m9999999k`	2155.00	2156.32	2165.48	2234.62	2718.56	6188.78	27875.09				

Figure 4.3: Memory Allocation [MB] of SPIN with Various Compile-time and Run-time Options.

Run	2	3	4	5	6	7	8	9	10	11	12
`-S 0`	15.60	16.20	17.28	20.10	31.85	81.98	307.95	1287.21	5489.53	23505.53	
`-S 1`	15.56	16.15	17.21	19.96	31.78	81.93	307.92	1287.12	5489.50	23505.54	
`-S 2`	15.56	16.12	17.23	19.96	31.84	82.21	309.35	1292.79	5515.70	23612.65	
`-S 0 -A`	15.57	16.20	17.43	20.87	34.81	95.32	358.09	1499.48	6359.03	27396.53	
`-S 1 -A`	15.53	16.14	17.39	20.85	34.79	95.29	358.09	1499.50	6358.95	27396.57	
`-S 2 -A`	15.50	16.17	17.40	20.78	34.82	95.60	359.43	1505.17	6385.17	27503.62	
`-S 2 -C`	15.50	16.18	17.40	20.78	34.84	95.56	359.48	1505.15	6385.09	27503.65	
`-S 2 -Z`	143.53	144.18	145.31	148.50	161.70	217.90	460.73	1509.68	5939.50	24886.25	89436.68 (‡)
`-S 2 -n 0`	15.59	16.15	17.21	20.09	31.95	82.26	309.29	1292.82	5515.71	23612.68	
`-S 2 -n 1`	15.46	16.06	17.12	19.92	31.75	82.14	309.17	1292.78	5515.57	23612.48	
`-S 2 -n 2`	15.56	16.20	17.25	20.04	31.92	82.25	309.40	1292.89	5515.70	23612.59	
`-S 2 -n 3`	15.57	16.14	17.25	19.96	31.90	82.26	309.37	1292.87	5515.75	23612.68	
`-S 2 -n 4`	15.51	16.15	17.23	19.95	31.84	82.25	309.32	1292.89	5515.70	23612.62	

Figure 4.4: Memory Allocation [MB] of UPPAAL with Various Run-time Options.

5. Evaluation and Conclusion

In all the samples, time-and memory consumption follow an exponential slope (after some offset). This follows to state-space increment and is hardly surprising.

Comparing the performance of UPPAAL and SPIN as used as a backend *for this specific example*, the following observations can be made.

1. **UPPAAL does perform significantly better than SPIN (here).**

   With respect to run-time, UPPAAL is ≈ 24 times faster when comparing best-to-best, without use of compiler optimizations (`-O2, -O3`) for SPIN the factor would be ≈ 53.[1]

   With respect to memory allocation, UPPAAL uses only $\approx 1/87$.[2]

   With all combinations, UPPAAL was able to process $N = 11$ clients, while SPIN ran out of memory after at most 9 clients.[3]

2. **Compile-Time optimization gives some time-improvements for SPIN.**

   The SPIN verifier is ≈ 2.4 times faster when compiled with optimization. It is mildly surprising that `-O2` actually performs a bit *better* than `-O3`. This is—of course—dependent on the used `C` compiler (see Appendix C.2.1).

   Note that the optimizations do essentially not affect the *memory* consumption. Memory remains the limiting factor.

3. **No Run-Time option gives significant improvements (here).**

   The best and worst were all within a factor of 2.

   Apparently the symmetry of the N client machines cannot be exploited in a significant way (by the tried options).

[1] The factor is derived as

$$\frac{\Sigma_{N=2..8}(\text{time of successful SPIN runs})/(\text{number of successful SPIN runs})}{\Sigma_{N=2..8}(\text{time of successful UPPAAL runs})/(\text{number of successful UPPAAL runs})}$$

[2] This comparison is somewhat unfair, since run-time option `-m9999999k` forces SPIN to allocate a big hash-table even for small value of N. Adjusting this option to "just fit" the model size would yield better result for SPIN here.

[3] $N = 9$ completes for the SPIN execution with run-time option `-m999k`; the outcome is unreliable due to limited search depth. Reliable computations are possible up to 8 clients.

A positive observation on the side is that all successful runs completed within the hour.[4] This means that (for 24GB of main memory) there is an acceptable time, after which we can stop waiting.

Open Questions. This investigation does not answer *why* the SPIN backend performs significantly worse than the UPPAAL backend. Maybe UPPAAL profits substantially from the fact that only the reachable states have to be allocated at all, while SPIN does provide (hash-compressed) memory for the full state space.

Possibly, the GTL-translation to SPIN/PROMELA (see Appendix B.2) leaves room for improvement.

Acknowledgments. Thanks go to Alexandre David for suggesting to consider compiler optimization options for the SPIN measurement.

[4]The unsuccessful (aborted) runs are not displayed in the plots or tables. The longest observed run lasted for 4.4 hours, before it got killed due to out-of-memory condition.

Bibliography

[1] Informatik Consulting Systems AG and Institut für Theoretische Information, Technische Universität Braunschweig and Verified Systems International GmbH: *VerSyKo — Verifikation von Systemen synchroner Software-Komponenten Contract Specification and Domain Specific Modeling Language for GALS Systems An Approach to System Validation.* available at http://www.iti.cs.tu-bs.de/~milius/research/konzeptpapier.pdf

[2] Gerard J. Holzmann: *The SPIN MODEL CHECKER – Primer and Reference Manual.* Addison-Wesley 2003, ISBN 0-321-22862-6. See also: http://spinroot.com/spin/whatispin.html

[3] Gerd Behrmann and Alexandre David and Kim Guldstrand Larsen and Paul Pettersson and Wang Yi: *Developing UPPAAL over 15 years.* Software: Practice and Experience, Vol 41(2), 2011, pages 133-142. See also: http://www.uppaal.com

[4] Nancy A. Lynch: *Distributed Algorithms.* Morgan Kaufmann, 1996. ISBN 1-55860-348-4.

Appendix A. Scripts and Utilities

A.1 Script: gen-mutex-gtl.awk

This script was used to generate the mutex_<N>.gtl files.

FILE: utils/gen-mutex-gtl.awk

```
#!/bin/bash
{ exec awk -f "`dirname $0`/gen-mutex-gtl.awk" "$1"; }
## ------------------------------------------------------------------------
## $Id: gen-mutex-gtl.awk 1769 2012-01-31 10:55:35Z moeller $
## $URL: svn://theo.iti.cs.tu-bs.de/studi_svn/VerSyKo/Uppaal/Mutex-Benchmark/gen-mutex-gtl.awk $
## ------------------------------------------------------------------------

function fprint_client(filename){
    print "model[none] client() {" >> filename
    print "  input bool proceed;" >> filename
    print "  output enum { nc, acq, cs, rel } st;" >> filename
    print "  init st 'nc;" >> filename
    print "  automaton {" >> filename
    print "    init state nc {" >> filename
    print "      st = 'nc;" >> filename
    print "      transition acq;" >> filename
    print "      transition nc;" >> filename
    print "    }" >> filename
    print "    state acq {" >> filename
    print "      st = 'acq;" >> filename
    print "      transition[proceed] cs;" >> filename
    print "      transition[!proceed] acq;" >> filename
    print "    }" >> filename
    print "    state cs {" >> filename
    print "      st = 'cs;" >> filename
    print "      transition rel;" >> filename
    print "      transition cs;" >> filename
    print "    }" >> filename
    print "    state rel {" >> filename
    print "      st = 'rel;" >> filename
    print "      transition nc;" >> filename
    print "    }" >> filename
    print "  };" >> filename
    print "}" >> filename
    print "" >> filename
}

function fprint_false_array(filename, N, the_true_one){
    printf "[" >> filename;
```

```
40      for( i = 0; i < N; i++ ){
41      if (i > 0) { printf "," >> filename; }
42      if (i == the_true_one){
43          printf "true" >>filename;
44      }
45      else {
46          printf "false" >>filename;
47      }
48      }
49      printf "]" >> filename;
50  }
51
52  function fprint_server(filename, N){
53      print "model[none] server() {" >> filename;
54      print "  input enum { nc, acq, cs, rel }^" N " procstates;" >>filename;
55      print "  output bool^" N " procouts;" >>filename;
56      printf "  init procouts " >>filename;
57      fprint_false_array(filename,N,-1);
58      printf ";\n" >>filename;
59      print "    always false " >> filename;
60      for(grant = 0; grant < N; grant++){
61      printf "      or (" >> filename;
62      for (j = 0; j < grant; j++){
63          printf "procstates[" j "] != 'acq and " >> filename;
64      }
65      printf "procstates[" grant "] = 'acq and " >> filename;
66      for (j = 0; j < N; j++){
67          if ( j != grant ){
68          printf "procstates[" j "] != 'cs and " >> filename;
69          }
70      }
71      printf "procouts = " >> filename;
72      fprint_false_array(filename,N,grant);
73      printf ")\n" >> filename;
74  #    (procstates[0] = 'acq and procstates[1] != 'cs and procstates[2] != 'cs and
        ↪ procouts = [true,false,false])
75  #          or (procstates[0] != 'acq and procstates[1] = 'acq and procstates[0]
        ↪ != 'cs and procstates[2] != 'cs and procouts = [false,true,false])
76  #          or (procstates[0] != 'acq and procstates[1] != 'acq and procstates[2]
        ↪ = 'acq and procstates[0] != 'cs and procstates[1] != 'cs and procouts = [
        ↪ false,false,true])
77      }
78      printf "    or (procouts = " >> filename;
79      fprint_false_array(filename,N,-1);
80      printf ");\n" >> filename;
81
82      print "}" >>filename;
83  }
84
85
86  BEGIN {
87      N = ARGV[1] + 0;
88
89      if( N < 1 ) {
90      print "usage: gen-mutex-gtl.awk <N>\n"
```

```
91      exit 1;
92      }
93      print "## generating for: N = " N;
94
95      filename = "./mutex_" N ".gtl";
96      print "## output gtl file = " filename;
97
98      print "// GENERATED BY gen-mutex-gtl.awk" > filename
99
100     fprint_client(filename);
101     fprint_server(filename, N);
102
103     print "\n// instances " >> filename
104     for ( i = 0; i < N; i++ ) {
105     print "instance client_c" i ";" >> filename
106     }
107     print "instance server_s;" >> filename
108
109     print "\n// connections " >> filename
110     for (i = 0; i < N; i++) {
111     print "connect c" i ".st s.procstates[" i "];" >> filename
112     }
113     for (i = 0; i < N; i++) {
114     print "connect s.procouts[" i "] c" i ".proceed;" >> filename
115     }
116     print "\n// safety claim" >> filename;
117     print "verify {" >> filename;
118     for (i = 0; i < N; i++){
119     printf "  always (c" i ".st = 'cs => !(false " >> filename;
120     for (j = 0; j < N; j++){
121         if ( i != j ){
122             printf "or c" j ".st = 'cs " >> filename;
123         }
124
125     }
126     printf "));\n" >> filename;
127     }
128     print "}" >> filename;
129     print "" >> filename
130
131     query_file = "./mutex_" N ".q";
132
133     print "## output query file = " query_file;
134     print "// GENERATED BY gen-mutex-gtl.awk" > query_file;
135     printf "A[] (not (false " >> query_file;
136     for (i = 0; i < N; i++) {
137     for (j = 0; j < i; j++) {
138         printf "|| (c" j ".l1 && c" i ".l1) " >> query_file;
139     }
140     }
141     printf " ))\n" >> query_file;
142
143     exit 0;
144 }
```

A.2 Script: `measure.bash`

This script was used to measure the time/memory consumption when executing the model-checking algorithm. Note that the system-utility

```
/usr/bin/time
```

is used.

FILE: `utils/measure.bash`

```
#!/bin/bash
## ----------------------------------------------------------------------
## $Id: measure.bash 1843 2012-02-06 09:18:34Z moeller $
## $URL: svn://theo.iti.cs.tu-bs.de/studi_svn/VerSyKo/Uppaal/Mutex-Benchmark/measure.bash $
## ----------------------------------------------------------------------
Revision=Revision
Empty=""
echo "## Measured with measure.bash [$Revision: 1843 $Empty]"
/usr/bin/time -f "\nreal %e\nuser %U\nsys %S\nMemory %M (KB)\nSwaps %W\n"  $@ 2>&1;

## this would be bash-builtin
##time $@ 2>&1;
```

The used `/usr/bin/time` - Version and Manpage.

FILE: `brief_info_time.txt`

```


[oli@bull measured.bull]$ /usr/bin/time --version
GNU time 1.7
man time
TIME(1)                                                                 TIME(1)



NAME
       time - time a simple command or give resource usage

SYNOPSIS
       time [options] command [arguments...]

DESCRIPTION
       The time command runs the specified program command with the given
       arguments. When command finishes, time writes a message to standard
       error giving timing statistics about this program run. These statis-
       tics consist of (i) the elapsed real time between invocation and termi-
       nation, (ii) the user CPU time (the sum of the tms_utime and tms_cutime
       values in a struct tms as returned by times(2)), and (iii) the system
```

```
23        CPU  time  (the  sum of the tms_stime and tms_cstime values in a struct
24        tms as returned by times(2)).
25
26        Note: some shells (e.g., bash(1)) have a  built-in  time  command  that
27        provides less functionality than the command described here.  To access
28        the real command, you may need to specify its pathname (something  like
29        /usr/bin/time).
30
31  OPTION
32        -p      When in the POSIX locale, use the precise traditional format
33                     "real %f\nuser %f\nsys %f\n"
34                (with  numbers  in  seconds) where the number of decimals in the
35                output for %f is unspecified but is sufficient  to  express  the
36                clock tick accuracy, and at least one.
37
38  ENVIRONMENT
39        The  variables LANG, LC_ALL, LC_CTYPE, LC_MESSAGES, LC_NUMERIC, NLSPATH
40        and PATH are used. The last one to search for command.  The  remaining
41        ones for the text and formatting of the output.
42
43  EXIT STATUS
44        If  command was invoked, the exit status is that of command.  Otherwise
45        it is 127 if command could not be found, 126 if it could be  found  but
46        could  not  be  invoked, and some other non-zero value (1-125) if some-
47        thing else went wrong.
48
49  SEE ALSO
50        times(2)
51
52
53
54  GNU VERSION
55        Below a description of the GNU 1.7 version of time.  Disregarding  the
56        name  of  the  utility, GNU makes it output lots of useful information,
57        not only about time used, but also on other resources like memory,  I/O
58        and  IPC calls (where available).  The output is formatted using a for-
59        mat string that can be specified using the -f option or the TIME  envi-
60        ronment variable.
61
62        The default format string is
63           %Uuser %Ssystem %Eelapsed %PCPU (%Xtext+%Ddata %Mmax)k
64           %Iinputs+%Ooutputs (%Fmajor+%Rminor)pagefaults %Wswaps
65
66        When the -p option is given the (portable) output format
67           real %e
68           user %U
69           sys %S
70        is used.
71
72     The format string
73        The format is interpreted in the usual printf-like way.  Ordinary char-
74        acters are directly copied, tab,  newline  and  backslash  are  escaped
75        using \t, \n and \\, a percent sign is represented by %%, and otherwise
76        % indicates a conversion. The program time will always add  a  trailing
77        newline  itself.  The conversions follow. All of those used by tcsh(1)
```

```
78          are supported.
79
80          Time
81
82          %E      Elapsed real time (in [hours:]minutes:seconds).
83
84          %e      (Not in tcsh.) Elapsed real time (in seconds).
85
86          %S      Total number of CPU-seconds that the process spent in kernel
87                  mode.
88
89          %U      Total number of CPU-seconds that the process spent in user mode.
90
91          %P      Percentage of the CPU that this job got, computed as (%U + %S) /
92                  %E.
93
94          Memory
95
96          %M      Maximum resident set size of the process during its lifetime, in
97                  Kbytes.
98
99          %t      (Not in tcsh.) Average resident set size of the process, in
100                 Kbytes.
101
102         %K      Average total (data+stack+text) memory use of the process, in
103                 Kbytes.
104
105         %D      Average size of the process's unshared data area, in Kbytes.
106
107         %p      (Not in tcsh.) Average size of the process's unshared stack
108                 space, in Kbytes.
109
110         %X      Average size of the process's shared text space, in Kbytes.
111
112         %Z      (Not in tcsh.) System's page size, in bytes. This is a per-sys-
113                 tem constant, but varies between systems.
114
115         %F      Number of major page faults that occurred while the process was
116                 running. These are faults where the page has to be read in from
117                 disk.
118
119         %R      Number of minor, or recoverable, page faults. These are faults
120                 for pages that are not valid but which have not yet been claimed
121                 by other virtual pages. Thus the data in the page is still
122                 valid but the system tables must be updated.
123
124         %W      Number of times the process was swapped out of main memory.
125
126         %c      Number of times the process was context-switched involuntarily
127                 (because the time slice expired).
128
129         %w      Number of waits: times that the program was context-switched
130                 voluntarily, for instance while waiting for an I/O operation to
131                 complete.
132
```

```
133         I/O
134
135         %I      Number of file system inputs by the process.
136
137         %O      Number of file system outputs by the process.
138
139         %r      Number of socket messages received by the process.
140
141         %s      Number of socket messages sent by the process.
142
143         %k      Number of signals delivered to the process.
144
145         %C      (Not in tcsh.) Name and command line arguments of the command
146                 being timed.
147
148         %x      (Not in tcsh.) Exit status of the command.
149
150 GNU OPTIONS
151         -f FORMAT, --format=FORMAT
152                 Specify output format, possibly overriding the format specified
153                 in the environment variable TIME.
154
155         -p, --portability
156                 Use the portable output format.
157
158         -o FILE, --output=FILE
159                Do not send the results to stderr, but overwrite the specified
160                file.
161
162         -a, --append
163                 (Used together with -o.) Do not overwrite but append.
164
165         -v, --verbose
166                 Give very verbose output about all the program knows about.
167
168 GNU STANDARD OPTIONS
169         --help Print a usage message on standard output and exit successfully.
170
171         -V, --version
172                 Print version information on standard output, then exit success-
173                 fully.
174
175         --      Terminate option list.
176
177 BUGS
178         Not all resources are measured by all versions of Unix, so some of the
179         values might be reported as zero. The present selection was mostly
180         inspired by the data provided by 4.2 or 4.3BSD.
181
182         GNU time version 1.7 is not yet localized. Thus, it does not implement
183         the POSIX requirements.
184
185         The environment variable TIME was badly chosen. It is not unusual for
186         systems like autoconf or make to use environment variables with the
187         name of a utility to override the utility to be used. Uses like MORE or
```

```
188         TIME for options to programs (instead of program pathnames) tend to
189         lead to difficulties.
190
191         It seems unfortunate that -o overwrites instead of appends. (That is,
192         the -a option should be the default.)
193
194         Mail suggestions and bug reports for GNU time to
195         bug-utils@prep.ai.mit.edu
196         Please include the version of time, which you can get by running
197         time --version
198         and the operating system and C compiler you used.
199
200 SEE ALSO
201         tcsh(1), times(2), wait3(2)
202
203 AUTHOR
204         David Keppel
205                 Original version
206
207         David MacKenzie
208                 POSIXization, autoconfiscation, GNU getoptization, docu-
209                 mentation, other bug fixes and improvements.
210
211         Arne Henrik Juul
212                 Helped with portability
213
214         Francois Pinard
215                 Helped with portability
216
217
218
219                                    2000-12-11                          TIME(1)
```

A.3 Utility: `plot.Makefile`

This Makefile was used to generate the plots.

FILE: `utils/plot.Makefile`

```
1  ## ----------------------------------------------------------------------------
2  ## $RCSfile: plot.Makefile,v $     $Revision: 1.3 $
3  ## $Header: /home/repository/VVTCVS/CVS/versyko/WHITEPAPERS/benchmark-mutex/↩
   ↪utils/plot.Makefile,v 1.3 2012/02/09 10:23:30 oli Exp $
4  ## ----------------------------------------------------------------------------
5
6  ROOT:=$(shell readlink -f `pwd`/../..)
7  BASENAME:=$(shell pwd | sed 's+^.*/[US]MEASURE.++g')
8  BASEPDFNAME:=$(shell echo $(BASENAME) | sed 's+[.]+__+g')
9  DATS:=time.dat memory.dat
10 PLOTS:=plot_time_$(BASEPDFNAME).pdf plot_memory_$(BASEPDFNAME).pdf
11
12 TOOL_CHAR:=$(shell pwd | sed 's+^.*\([SU]\)MEASURE[.].*+\1+g')
```

```
13 ifeq ($(TOOL_CHAR),U)
14 TOOL:=Uppaal
15 TITLE:=Uppaal64-4.1.7
16 COLOR:=blue
17 OPTIONS:=$(shell echo $(BASENAME) | sed 's/_/␣/g')
18 else
19 TOOL:=Spin
20 TITLE:=Spin-6.1.0
21 COLOR:=red
22 OPTIONS:=Compile= $(shell echo $(BASENAME) | cut -d'.' -f 1| sed 's/_/␣/g') Run↩
       ↪= $(shell echo $(BASENAME) | cut -d'.' -f 2| sed 's/_/␣/g')
23 endif
24
25 clean:
26     rm -f *.gpl
27     rm -f *.ps
28     rm -f *.pdf
29     rm -f *.dat
30
31 %_$(BASENAME).gpl: $(ROOT)/%_COMMON.gpl
32     cat $(<) \
33     | sed 's/@BASENAME@/$(BASENAME)/g' \
34     | sed 's/@COLOR@/$(COLOR)/g' \
35     | sed 's/@TITLE@/$(TITLE)/g' \
36     | sed 's/@OPTIONS@/$(OPTIONS)/g' \
37     > $(@)
38
39 %.ps: %.gpl
40     gnuplot $(<)
41
42 %$(BASEPDFNAME).pdf: %$(BASENAME).ps
43     ps2pdf $(<) $(@)
44
45 %.bb: %.ps
46     grep BoundingBox $(<) >$(@)
47
48 time.dat: $(wildcard *log)
49     $(ROOT)/utils/get_time.bash >$(@)
50
51 time_real.dat: $(wildcard *log)
52     $(ROOT)/utils/get_time.bash real >$(@)
53
54 memory.dat: $(wildcard *log)
55     $(ROOT)/utils/get_memory_MB.bash >$(@)
56
57 ## ----------------------------------------------------------------------
58
59 all: $(DATS) $(PLOTS)
60
61
62 install: $(PLOTS)
63     cp $(PLOTS) $(ROOT)/PLOTS/
64
65
66 ## ----------------------------------------------------------------------
```

```
67 .PHONEY: debug show
68
69 show: all
70     xpdf -z page plot_time_$(BASENAME).pdf &
71     xpdf -z page plot_memory_$(BASENAME).pdf &
72
73 debug:
74     @echo ROOT=$(ROOT)
75     @echo BASENAME=$(BASENAME)
76     @echo BASEPDFNAME=$(BASEPDFNAME)
77     @echo TOOL_CHAR=$(TOOL_CHAR)
78     @echo TOOL=$(TOOL)
79     @echo TITLE=$(TITLE)
80     @echo COLOR=$(COLOR)
81
82 ## ------------------------------------------------------------------------------
```

Appendix B. Generated Code

B.1 Script-generated GTL models

The following GTL-files were generated by `gen-mutex-gtl.awk`, see Appendix A.1.

B.1.1 GTL code for Mutex with 2 Clients

FILE: `mutex_2.gtl`

```
// GENERATED BY gen-mutex-gtl.awk
model[none] client() {
  input bool proceed;
  output enum { nc, acq, cs, rel } st;
  init st 'nc;
  automaton {
    init state nc {
      st = 'nc;
      transition acq;
      transition nc;
    }
    state acq {
      st = 'acq;
      transition[proceed] cs;
      transition[!proceed] acq;
    }
    state cs {
      st = 'cs;
      transition rel;
      transition cs;
    }
    state rel {
      st = 'rel;
      transition nc;
    }
  };
}

model[none] server() {
  input enum { nc, acq, cs, rel }^2 procstates;
  output bool^2 procouts;
  init procouts [false,false];
   always false
    or (procstates[0] = 'acq and procstates[1] != 'cs and procouts = [true,false])
    or (procstates[0] != 'acq and procstates[1] = 'acq and procstates[0] != 'cs and procouts = [false,true])
    or (procouts = [false,false]);
```

```
}

// instances
instance client c0;
instance client c1;
instance server s;

// connections
connect c0.st s.procstates[0];
connect c1.st s.procstates[1];
connect s.procouts[0] c0.proceed;
connect s.procouts[1] c1.proceed;

// safety claim
verify {
  always (c0.st = 'cs => !(false or c1.st = 'cs ));
  always (c1.st = 'cs => !(false or c0.st = 'cs ));
}
```

B.1.2 GTL code for Mutex with 3 Clients

FILE: mutex_3.gtl

```
// GENERATED BY gen-mutex-gtl.awk
model[none] client() {
  input bool proceed;
  output enum { nc, acq, cs, rel } st;
  init st 'nc;
  automaton {
    init state nc {
      st = 'nc;
      transition acq;
      transition nc;
    }
    state acq {
      st = 'acq;
      transition[proceed] cs;
      transition[!proceed] acq;
    }
    state cs {
      st = 'cs;
      transition rel;
      transition cs;
    }
    state rel {
      st = 'rel;
      transition nc;
    }
  };
}

model[none] server() {
  input enum { nc, acq, cs, rel }^3 procstates;
  output bool^3 procouts;
```

```
32    init procouts [false,false,false];
33     always false
34      or (procstates[0] = 'acq and procstates[1] != 'cs and procstates[2] != 'cs ↩
      ↪and procouts = [true,false,false])
35      or (procstates[0] != 'acq and procstates[1] = 'acq and procstates[0] != 'cs ↩
      ↪and procstates[2] != 'cs and procouts = [false,true,false])
36      or (procstates[0] != 'acq and procstates[1] != 'acq and procstates[2] = 'acq↩
      ↪ and procstates[0] != 'cs and procstates[1] != 'cs and procouts = [false,↩
      ↪false,true])
37      or (procouts = [false,false,false]);
38  }
39
40  // instances
41  instance client c0;
42  instance client c1;
43  instance client c2;
44  instance server s;
45
46  // connections
47  connect c0.st s.procstates[0];
48  connect c1.st s.procstates[1];
49  connect c2.st s.procstates[2];
50  connect s.procouts[0] c0.proceed;
51  connect s.procouts[1] c1.proceed;
52  connect s.procouts[2] c2.proceed;
53
54  // safety claim
55  verify {
56    always (c0.st = 'cs => !(false or c1.st = 'cs or c2.st = 'cs ));
57    always (c1.st = 'cs => !(false or c0.st = 'cs or c2.st = 'cs ));
58    always (c2.st = 'cs => !(false or c0.st = 'cs or c1.st = 'cs ));
59  }
```

B.2 SPIN/PROMELA Files Generated by the GTL Tool

The following PROMELA-files were generated by GTL tool, Version 0.1 (2012-01-09) via processing of `mutex_N.glt` (see B.1.1ff):

```
gtl  mutex_<N>.gtl
```

B.2.1 PROMELA Model: Mutex with 2 Clients

FILE: `mutex_2.pr`

```
1  bool c0_proceed_0 = false
2  int c0_st_0 = 0
3  bool c1_proceed_0 = false
4  int c1_st_0 = 0
5  int s_procstates_0_0 = 0
6  int s_procstates_1_0 = 0
7  int _count_c0, _count_c1, _count_s
8  hidden int _minimum
```

```
proctype c0() {
  if
    :: atomic {
         _count_c0 == 0;
         if
           :: s_procstates_0_0 = 1;
              c0_st_0 = s_procstates_0_0
         fi;
         printf("TRANSITION␣c0␣2␣0\n");
         d_step {
           _count_c0 = _count_c0 + 1;
           _minimum = _count_c0;
           if
             :: _count_c1 < _minimum;
                _minimum = _count_c1
             :: else
           fi;
           if
             :: _count_s < _minimum;
                _minimum = _count_s
             :: else
           fi;
           _count_c0 = _count_c0 - _minimum;
           _count_c1 = _count_c1 - _minimum;
           _count_s = _count_s - _minimum
         };
         goto st0
       }
    :: atomic {
         _count_c0 == 0;
         if
           :: s_procstates_0_0 = 0;
              c0_st_0 = s_procstates_0_0
         fi;
         printf("TRANSITION␣c0␣2␣1\n");
         d_step {
           _count_c0 = _count_c0 + 1;
           _minimum = _count_c0;
           if
             :: _count_c1 < _minimum;
                _minimum = _count_c1
             :: else
           fi;
           if
             :: _count_s < _minimum;
                _minimum = _count_s
             :: else
           fi;
           _count_c0 = _count_c0 - _minimum;
           _count_c1 = _count_c1 - _minimum;
           _count_s = _count_s - _minimum
         };
         goto st2
       }
  fi;
```

```
64  st0:
65    if
66      :: atomic {
67           _count_c0 == 0 && c0_proceed_0;
68           if
69             :: s_procstates_0_0 = 2;
70                c0_st_0 = s_procstates_0_0
71           fi;
72           printf("TRANSITION␣c0␣0␣0\n");
73           d_step {
74             _count_c0 = _count_c0 + 1;
75             _minimum = _count_c0;
76             if
77               :: _count_c1 < _minimum;
78                  _minimum = _count_c1
79               :: else
80             fi;
81             if
82               :: _count_s < _minimum;
83                  _minimum = _count_s
84               :: else
85             fi;
86             _count_c0 = _count_c0 - _minimum;
87             _count_c1 = _count_c1 - _minimum;
88             _count_s = _count_s - _minimum
89           };
90           goto st1
91         }
92      :: atomic {
93           _count_c0 == 0 && !(c0_proceed_0);
94           if
95             :: s_procstates_0_0 = 1;
96                c0_st_0 = s_procstates_0_0
97           fi;
98           printf("TRANSITION␣c0␣0␣1\n");
99           d_step {
100            _count_c0 = _count_c0 + 1;
101            _minimum = _count_c0;
102            if
103              :: _count_c1 < _minimum;
104                 _minimum = _count_c1
105              :: else
106            fi;
107            if
108              :: _count_s < _minimum;
109                 _minimum = _count_s
110              :: else
111            fi;
112            _count_c0 = _count_c0 - _minimum;
113            _count_c1 = _count_c1 - _minimum;
114            _count_s = _count_s - _minimum
115          };
116          goto st0
117        }
118   fi;
```

```
119   st1:
120     if
121       :: atomic {
122            _count_c0 == 0;
123            if
124              :: s_procstates_0_0 = 3;
125                 c0_st_0 = s_procstates_0_0
126            fi;
127            printf("TRANSITION␣c0␣1␣0\n");
128            d_step {
129              _count_c0 = _count_c0 + 1;
130              _minimum = _count_c0;
131              if
132                :: _count_c1 < _minimum;
133                   _minimum = _count_c1
134                :: else
135              fi;
136              if
137                :: _count_s < _minimum;
138                   _minimum = _count_s
139                :: else
140              fi;
141              _count_c0 = _count_c0 - _minimum;
142              _count_c1 = _count_c1 - _minimum;
143              _count_s = _count_s - _minimum
144            };
145            goto st3
146          }
147       :: atomic {
148            _count_c0 == 0;
149            if
150              :: s_procstates_0_0 = 2;
151                 c0_st_0 = s_procstates_0_0
152            fi;
153            printf("TRANSITION␣c0␣1␣1\n");
154            d_step {
155              _count_c0 = _count_c0 + 1;
156              _minimum = _count_c0;
157              if
158                :: _count_c1 < _minimum;
159                   _minimum = _count_c1
160                :: else
161              fi;
162              if
163                :: _count_s < _minimum;
164                   _minimum = _count_s
165                :: else
166              fi;
167              _count_c0 = _count_c0 - _minimum;
168              _count_c1 = _count_c1 - _minimum;
169              _count_s = _count_s - _minimum
170            };
171            goto st1
172          }
173     fi;
```

```
174   st2:
175     if
176       :: atomic {
177            _count_c0 == 0;
178            if
179              :: s_procstates_0_0 = 1;
180                 c0_st_0 = s_procstates_0_0
181            fi;
182            printf("TRANSITION␣c0␣2␣0\n");
183            d_step {
184              _count_c0 = _count_c0 + 1;
185              _minimum = _count_c0;
186              if
187                :: _count_c1 < _minimum;
188                   _minimum = _count_c1
189                :: else
190              fi;
191              if
192                :: _count_s < _minimum;
193                   _minimum = _count_s
194                :: else
195              fi;
196              _count_c0 = _count_c0 - _minimum;
197              _count_c1 = _count_c1 - _minimum;
198              _count_s = _count_s - _minimum
199            };
200            goto st0
201          }
202       :: atomic {
203            _count_c0 == 0;
204            if
205              :: s_procstates_0_0 = 0;
206                 c0_st_0 = s_procstates_0_0
207            fi;
208            printf("TRANSITION␣c0␣2␣1\n");
209            d_step {
210              _count_c0 = _count_c0 + 1;
211              _minimum = _count_c0;
212              if
213                :: _count_c1 < _minimum;
214                   _minimum = _count_c1
215                :: else
216              fi;
217              if
218                :: _count_s < _minimum;
219                   _minimum = _count_s
220                :: else
221              fi;
222              _count_c0 = _count_c0 - _minimum;
223              _count_c1 = _count_c1 - _minimum;
224              _count_s = _count_s - _minimum
225            };
226            goto st2
227          }
228     fi;
```

```
229   st3:
230     if
231       :: atomic {
232            _count_c0 == 0;
233            if
234              :: s_procstates_0_0 = 0;
235                 c0_st_0 = s_procstates_0_0
236            fi;
237            printf("TRANSITION␣c0␣3␣0\n");
238            d_step {
239              _count_c0 = _count_c0 + 1;
240              _minimum = _count_c0;
241              if
242                :: _count_c1 < _minimum;
243                   _minimum = _count_c1
244                :: else
245              fi;
246              if
247                :: _count_s < _minimum;
248                   _minimum = _count_s
249                :: else
250              fi;
251              _count_c0 = _count_c0 - _minimum;
252              _count_c1 = _count_c1 - _minimum;
253              _count_s = _count_s - _minimum
254            };
255            goto st2
256          }
257     fi
258 }
259 proctype c1() {
260   if
261     :: atomic {
262          _count_c1 == 0;
263          if
264            :: s_procstates_1_0 = 1;
265               c1_st_0 = s_procstates_1_0
266          fi;
267          printf("TRANSITION␣c1␣2␣0\n");
268          d_step {
269            _count_c1 = _count_c1 + 1;
270            _minimum = _count_c0;
271            if
272              :: _count_c1 < _minimum;
273                 _minimum = _count_c1
274              :: else
275            fi;
276            if
277              :: _count_s < _minimum;
278                 _minimum = _count_s
279              :: else
280            fi;
281            _count_c0 = _count_c0 - _minimum;
282            _count_c1 = _count_c1 - _minimum;
283            _count_s = _count_s - _minimum
```

```
284           };
285           goto st0
286         }
287       :: atomic {
288           _count_c1 == 0;
289           if
290             :: s_procstates_1_0 = 0;
291                c1_st_0 = s_procstates_1_0
292           fi;
293           printf("TRANSITION␣c1␣2␣1\n");
294           d_step {
295             _count_c1 = _count_c1 + 1;
296             _minimum = _count_c0;
297             if
298               :: _count_c1 < _minimum;
299                  _minimum = _count_c1
300               :: else
301             fi;
302             if
303               :: _count_s < _minimum;
304                  _minimum = _count_s
305               :: else
306             fi;
307             _count_c0 = _count_c0 - _minimum;
308             _count_c1 = _count_c1 - _minimum;
309             _count_s = _count_s - _minimum
310           };
311           goto st2
312         }
313     fi;
314     st0:
315       if
316         :: atomic {
317             _count_c1 == 0 && c1_proceed_0;
318             if
319               :: s_procstates_1_0 = 2;
320                  c1_st_0 = s_procstates_1_0
321             fi;
322             printf("TRANSITION␣c1␣0␣0\n");
323             d_step {
324               _count_c1 = _count_c1 + 1;
325               _minimum = _count_c0;
326               if
327                 :: _count_c1 < _minimum;
328                    _minimum = _count_c1
329                 :: else
330               fi;
331               if
332                 :: _count_s < _minimum;
333                    _minimum = _count_s
334                 :: else
335               fi;
336               _count_c0 = _count_c0 - _minimum;
337               _count_c1 = _count_c1 - _minimum;
338               _count_s = _count_s - _minimum
```

```
339             };
340             goto st1
341           }
342         :: atomic {
343             _count_c1 == 0 && !(c1_proceed_0);
344             if
345               :: s_procstates_1_0 = 1;
346                  c1_st_0 = s_procstates_1_0
347             fi;
348             printf("TRANSITION␣c1␣0␣1\n");
349             d_step {
350               _count_c1 = _count_c1 + 1;
351               _minimum = _count_c0;
352               if
353                 :: _count_c1 < _minimum;
354                    _minimum = _count_c1
355                 :: else
356               fi;
357               if
358                 :: _count_s < _minimum;
359                    _minimum = _count_s
360                 :: else
361               fi;
362               _count_c0 = _count_c0 - _minimum;
363               _count_c1 = _count_c1 - _minimum;
364               _count_s = _count_s - _minimum
365             };
366             goto st0
367           }
368     fi;
369   st1:
370     if
371       :: atomic {
372             _count_c1 == 0;
373             if
374               :: s_procstates_1_0 = 3;
375                  c1_st_0 = s_procstates_1_0
376             fi;
377             printf("TRANSITION␣c1␣1␣0\n");
378             d_step {
379               _count_c1 = _count_c1 + 1;
380               _minimum = _count_c0;
381               if
382                 :: _count_c1 < _minimum;
383                    _minimum = _count_c1
384                 :: else
385               fi;
386               if
387                 :: _count_s < _minimum;
388                    _minimum = _count_s
389                 :: else
390               fi;
391               _count_c0 = _count_c0 - _minimum;
392               _count_c1 = _count_c1 - _minimum;
393               _count_s = _count_s - _minimum
```

```
394             };
395             goto st3
396           }
397         :: atomic {
398             _count_c1 == 0;
399             if
400               :: s_procstates_1_0 = 2;
401                  c1_st_0 = s_procstates_1_0
402             fi;
403             printf("TRANSITION␣c1␣1␣1\n");
404             d_step {
405               _count_c1 = _count_c1 + 1;
406               _minimum = _count_c0;
407               if
408                 :: _count_c1 < _minimum;
409                    _minimum = _count_c1
410                 :: else
411               fi;
412               if
413                 :: _count_s < _minimum;
414                    _minimum = _count_s
415                 :: else
416               fi;
417               _count_c0 = _count_c0 - _minimum;
418               _count_c1 = _count_c1 - _minimum;
419               _count_s = _count_s - _minimum
420             };
421             goto st1
422           }
423     fi;
424   st2:
425     if
426       :: atomic {
427             _count_c1 == 0;
428             if
429               :: s_procstates_1_0 = 1;
430                  c1_st_0 = s_procstates_1_0
431             fi;
432             printf("TRANSITION␣c1␣2␣0\n");
433             d_step {
434               _count_c1 = _count_c1 + 1;
435               _minimum = _count_c0;
436               if
437                 :: _count_c1 < _minimum;
438                    _minimum = _count_c1
439                 :: else
440               fi;
441               if
442                 :: _count_s < _minimum;
443                    _minimum = _count_s
444                 :: else
445               fi;
446               _count_c0 = _count_c0 - _minimum;
447               _count_c1 = _count_c1 - _minimum;
448               _count_s = _count_s - _minimum
```

```
449             };
450             goto st0
451           }
452         :: atomic {
453             _count_c1 == 0;
454             if
455               :: s_procstates_1_0 = 0;
456                  c1_st_0 = s_procstates_1_0
457             fi;
458             printf("TRANSITION␣c1␣2␣1\n");
459             d_step {
460               _count_c1 = _count_c1 + 1;
461               _minimum = _count_c0;
462               if
463                 :: _count_c1 < _minimum;
464                    _minimum = _count_c1
465                 :: else
466               fi;
467               if
468                 :: _count_s < _minimum;
469                    _minimum = _count_s
470                 :: else
471               fi;
472               _count_c0 = _count_c0 - _minimum;
473               _count_c1 = _count_c1 - _minimum;
474               _count_s = _count_s - _minimum
475             };
476             goto st2
477           }
478     fi;
479   st3:
480     if
481       :: atomic {
482             _count_c1 == 0;
483             if
484               :: s_procstates_1_0 = 0;
485                  c1_st_0 = s_procstates_1_0
486             fi;
487             printf("TRANSITION␣c1␣3␣0\n");
488             d_step {
489               _count_c1 = _count_c1 + 1;
490               _minimum = _count_c0;
491               if
492                 :: _count_c1 < _minimum;
493                    _minimum = _count_c1
494                 :: else
495               fi;
496               if
497                 :: _count_s < _minimum;
498                    _minimum = _count_s
499                 :: else
500               fi;
501               _count_c0 = _count_c0 - _minimum;
502               _count_c1 = _count_c1 - _minimum;
503               _count_s = _count_s - _minimum
```

```
504             };
505             goto st2
506           }
507       fi
508   }
509   proctype s() {
510     if
511       :: atomic {
512           _count_s == 0 && s_procstates_0_0 != 1 && s_procstates_1_0 == 1 && ↩
      ↪s_procstates_0_0 != 2;
513           c0_proceed_0 = false;
514           c1_proceed_0 = true;
515           printf("TRANSITION␣s␣0␣0\n");
516           d_step {
517             _count_s = _count_s + 1;
518             _minimum = _count_c0;
519             if
520               :: _count_c1 < _minimum;
521                  _minimum = _count_c1
522               :: else
523             fi;
524             if
525               :: _count_s < _minimum;
526                  _minimum = _count_s
527               :: else
528             fi;
529             _count_c0 = _count_c0 - _minimum;
530             _count_c1 = _count_c1 - _minimum;
531             _count_s = _count_s - _minimum
532           };
533           goto st0
534         }
535       :: atomic {
536           _count_s == 0;
537           c0_proceed_0 = false;
538           c1_proceed_0 = false;
539           printf("TRANSITION␣s␣0␣1\n");
540           d_step {
541             _count_s = _count_s + 1;
542             _minimum = _count_c0;
543             if
544               :: _count_c1 < _minimum;
545                  _minimum = _count_c1
546               :: else
547             fi;
548             if
549               :: _count_s < _minimum;
550                  _minimum = _count_s
551               :: else
552             fi;
553             _count_c0 = _count_c0 - _minimum;
554             _count_c1 = _count_c1 - _minimum;
555             _count_s = _count_s - _minimum
556           };
557           goto st0
```

```
558         }
559       :: atomic {
560           _count_s == 0 && s_procstates_0_0 == 1 && s_procstates_1_0 != 2;
561           c0_proceed_0 = true;
562           c1_proceed_0 = false;
563           printf("TRANSITION␣s␣0␣2\n");
564           d_step {
565             _count_s = _count_s + 1;
566             _minimum = _count_c0;
567             if
568               :: _count_c1 < _minimum;
569                  _minimum = _count_c1
570               :: else
571             fi;
572             if
573               :: _count_s < _minimum;
574                  _minimum = _count_s
575               :: else
576             fi;
577             _count_c0 = _count_c0 - _minimum;
578             _count_c1 = _count_c1 - _minimum;
579             _count_s = _count_s - _minimum
580           };
581           goto st0
582         }
583     fi;
584     st0:
585       if
586         :: atomic {
587             _count_s == 0 && s_procstates_0_0 != 1 && s_procstates_1_0 == 1 && ↩
      ↪s_procstates_0_0 != 2;
588             c0_proceed_0 = false;
589             c1_proceed_0 = true;
590             printf("TRANSITION␣s␣0␣0\n");
591             d_step {
592               _count_s = _count_s + 1;
593               _minimum = _count_c0;
594               if
595                 :: _count_c1 < _minimum;
596                    _minimum = _count_c1
597                 :: else
598               fi;
599               if
600                 :: _count_s < _minimum;
601                    _minimum = _count_s
602                 :: else
603               fi;
604               _count_c0 = _count_c0 - _minimum;
605               _count_c1 = _count_c1 - _minimum;
606               _count_s = _count_s - _minimum
607             };
608             goto st0
609           }
610         :: atomic {
611             _count_s == 0;
```

```
612             c0_proceed_0 = false;
613             c1_proceed_0 = false;
614             printf("TRANSITION␣s␣0␣1\n");
615             d_step {
616               _count_s = _count_s + 1;
617               _minimum = _count_c0;
618               if
619                 :: _count_c1 < _minimum;
620                    _minimum = _count_c1
621                 :: else
622               fi;
623               if
624                 :: _count_s < _minimum;
625                    _minimum = _count_s
626                 :: else
627               fi;
628               _count_c0 = _count_c0 - _minimum;
629               _count_c1 = _count_c1 - _minimum;
630               _count_s = _count_s - _minimum
631             };
632             goto st0
633           }
634         :: atomic {
635             _count_s == 0 && s_procstates_0_0 == 1 && s_procstates_1_0 != 2;
636             c0_proceed_0 = true;
637             c1_proceed_0 = false;
638             printf("TRANSITION␣s␣0␣2\n");
639             d_step {
640               _count_s = _count_s + 1;
641               _minimum = _count_c0;
642               if
643                 :: _count_c1 < _minimum;
644                    _minimum = _count_c1
645                 :: else
646               fi;
647               if
648                 :: _count_s < _minimum;
649                    _minimum = _count_s
650                 :: else
651               fi;
652               _count_c0 = _count_c0 - _minimum;
653               _count_c1 = _count_c1 - _minimum;
654               _count_s = _count_s - _minimum
655             };
656             goto st0
657           }
658       fi
659   }
660   init {
661     atomic {
662       skip;
663       run c0();
664       run c1();
665       run s()
666     }
```

```
667 }
668 never {
669   if
670     :: atomic {
671          goto st0
672        }
673     :: atomic {
674          c1_st_0 == 2 && c0_st_0 == 2;
675          goto st2
676        }
677     :: atomic {
678          goto st1
679        }
680     :: atomic {
681          c0_st_0 == 2 && c1_st_0 == 2;
682          goto st2
683        }
684   fi;
685   st0:
686     if
687       :: atomic {
688            goto st0
689          }
690       :: atomic {
691            c1_st_0 == 2 && c0_st_0 == 2;
692            goto st2
693          }
694     fi;
695   st1:
696     if
697       :: atomic {
698            goto st1
699          }
700       :: atomic {
701            c0_st_0 == 2 && c1_st_0 == 2;
702            goto st2
703          }
704     fi;
705   accept2:
706     st2:
707       if
708         :: atomic {
709              goto st2
710            }
711       fi
712 }
```

B.2.2 PROMELA Model: Mutex with 3 Clients

FILE: mutex_3.pr

```
1 bool c0_proceed_0 = false
2 int c0_st_0 = 0
3 bool c1_proceed_0 = false
```

```
int c1_st_0 = 0
bool c2_proceed_0 = false
int c2_st_0 = 0
int s_procstates_0_0 = 0
int s_procstates_1_0 = 0
int s_procstates_2_0 = 0
int _count_c0, _count_c1, _count_c2, _count_s
hidden int _minimum
proctype c0() {
  if
    :: atomic {
         _count_c0 == 0;
         if
           :: s_procstates_0_0 = 1;
              c0_st_0 = s_procstates_0_0
         fi;
         printf("TRANSITION␣c0␣2␣0\n");
         d_step {
           _count_c0 = _count_c0 + 1;
           _minimum = _count_c0;
           if
             :: _count_c1 < _minimum;
                _minimum = _count_c1
             :: else
           fi;
           if
             :: _count_c2 < _minimum;
                _minimum = _count_c2
             :: else
           fi;
           if
             :: _count_s < _minimum;
                _minimum = _count_s
             :: else
           fi;
           _count_c0 = _count_c0 - _minimum;
           _count_c1 = _count_c1 - _minimum;
           _count_c2 = _count_c2 - _minimum;
           _count_s = _count_s - _minimum
         };
         goto st0
       }
    :: atomic {
         _count_c0 == 0;
         if
           :: s_procstates_0_0 = 0;
              c0_st_0 = s_procstates_0_0
         fi;
         printf("TRANSITION␣c0␣2␣1\n");
         d_step {
           _count_c0 = _count_c0 + 1;
           _minimum = _count_c0;
           if
             :: _count_c1 < _minimum;
                _minimum = _count_c1
```

```
59              :: else
60            fi;
61            if
62              :: _count_c2 < _minimum;
63                 _minimum = _count_c2
64              :: else
65            fi;
66            if
67              :: _count_s < _minimum;
68                 _minimum = _count_s
69              :: else
70            fi;
71            _count_c0 = _count_c0 - _minimum;
72            _count_c1 = _count_c1 - _minimum;
73            _count_c2 = _count_c2 - _minimum;
74            _count_s = _count_s - _minimum
75          };
76          goto st2
77        }
78    fi;
79    st0:
80      if
81        :: atomic {
82             _count_c0 == 0 && c0_proceed_0;
83             if
84               :: s_procstates_0_0 = 2;
85                  c0_st_0 = s_procstates_0_0
86             fi;
87             printf("TRANSITION c0 0 0\n");
88             d_step {
89               _count_c0 = _count_c0 + 1;
90               _minimum = _count_c0;
91               if
92                 :: _count_c1 < _minimum;
93                    _minimum = _count_c1
94                 :: else
95               fi;
96               if
97                 :: _count_c2 < _minimum;
98                    _minimum = _count_c2
99                 :: else
100              fi;
101              if
102                :: _count_s < _minimum;
103                   _minimum = _count_s
104                :: else
105              fi;
106              _count_c0 = _count_c0 - _minimum;
107              _count_c1 = _count_c1 - _minimum;
108              _count_c2 = _count_c2 - _minimum;
109              _count_s = _count_s - _minimum
110            };
111            goto st1
112          }
113       :: atomic {
```

```
114             _count_c0 == 0 && !(c0_proceed_0);
115             if
116               :: s_procstates_0_0 = 1;
117                  c0_st_0 = s_procstates_0_0
118             fi;
119             printf("TRANSITION␣c0␣0␣1\n");
120             d_step {
121               _count_c0 = _count_c0 + 1;
122               _minimum = _count_c0;
123               if
124                 :: _count_c1 < _minimum;
125                    _minimum = _count_c1
126                 :: else
127               fi;
128               if
129                 :: _count_c2 < _minimum;
130                    _minimum = _count_c2
131                 :: else
132               fi;
133               if
134                 :: _count_s < _minimum;
135                    _minimum = _count_s
136                 :: else
137               fi;
138               _count_c0 = _count_c0 - _minimum;
139               _count_c1 = _count_c1 - _minimum;
140               _count_c2 = _count_c2 - _minimum;
141               _count_s = _count_s - _minimum
142             };
143             goto st0
144           }
145       fi;
146     st1:
147       if
148         :: atomic {
149             _count_c0 == 0;
150             if
151               :: s_procstates_0_0 = 3;
152                  c0_st_0 = s_procstates_0_0
153             fi;
154             printf("TRANSITION␣c0␣1␣0\n");
155             d_step {
156               _count_c0 = _count_c0 + 1;
157               _minimum = _count_c0;
158               if
159                 :: _count_c1 < _minimum;
160                    _minimum = _count_c1
161                 :: else
162               fi;
163               if
164                 :: _count_c2 < _minimum;
165                    _minimum = _count_c2
166                 :: else
167               fi;
168               if
```

```
169                 :: _count_s < _minimum;
170                    _minimum = _count_s
171                 :: else
172               fi;
173               _count_c0 = _count_c0 - _minimum;
174               _count_c1 = _count_c1 - _minimum;
175               _count_c2 = _count_c2 - _minimum;
176               _count_s = _count_s - _minimum
177             };
178             goto st3
179           }
180         :: atomic {
181             _count_c0 == 0;
182             if
183               :: s_procstates_0_0 = 2;
184                  c0_st_0 = s_procstates_0_0
185             fi;
186             printf("TRANSITION c0 1 1\n");
187             d_step {
188               _count_c0 = _count_c0 + 1;
189               _minimum = _count_c0;
190               if
191                 :: _count_c1 < _minimum;
192                    _minimum = _count_c1
193                 :: else
194               fi;
195               if
196                 :: _count_c2 < _minimum;
197                    _minimum = _count_c2
198                 :: else
199               fi;
200               if
201                 :: _count_s < _minimum;
202                    _minimum = _count_s
203                 :: else
204               fi;
205               _count_c0 = _count_c0 - _minimum;
206               _count_c1 = _count_c1 - _minimum;
207               _count_c2 = _count_c2 - _minimum;
208               _count_s = _count_s - _minimum
209             };
210             goto st1
211           }
212       fi;
213     st2:
214       if
215         :: atomic {
216             _count_c0 == 0;
217             if
218               :: s_procstates_0_0 = 1;
219                  c0_st_0 = s_procstates_0_0
220             fi;
221             printf("TRANSITION c0 2 0\n");
222             d_step {
223               _count_c0 = _count_c0 + 1;
```

```
224             _minimum = _count_c0;
225             if
226               :: _count_c1 < _minimum;
227                  _minimum = _count_c1
228               :: else
229             fi;
230             if
231               :: _count_c2 < _minimum;
232                  _minimum = _count_c2
233               :: else
234             fi;
235             if
236               :: _count_s < _minimum;
237                  _minimum = _count_s
238               :: else
239             fi;
240             _count_c0 = _count_c0 - _minimum;
241             _count_c1 = _count_c1 - _minimum;
242             _count_c2 = _count_c2 - _minimum;
243             _count_s = _count_s - _minimum
244           };
245           goto st0
246         }
247       :: atomic {
248           _count_c0 == 0;
249           if
250             :: s_procstates_0_0 = 0;
251                c0_st_0 = s_procstates_0_0
252           fi;
253           printf("TRANSITION␣c0␣2␣1\n");
254           d_step {
255             _count_c0 = _count_c0 + 1;
256             _minimum = _count_c0;
257             if
258               :: _count_c1 < _minimum;
259                  _minimum = _count_c1
260               :: else
261             fi;
262             if
263               :: _count_c2 < _minimum;
264                  _minimum = _count_c2
265               :: else
266             fi;
267             if
268               :: _count_s < _minimum;
269                  _minimum = _count_s
270               :: else
271             fi;
272             _count_c0 = _count_c0 - _minimum;
273             _count_c1 = _count_c1 - _minimum;
274             _count_c2 = _count_c2 - _minimum;
275             _count_s = _count_s - _minimum
276           };
277           goto st2
278         }
```

```
279     fi;
280   st3:
281     if
282       :: atomic {
283            _count_c0 == 0;
284            if
285              :: s_procstates_0_0 = 0;
286                 c0_st_0 = s_procstates_0_0
287            fi;
288            printf("TRANSITION␣c0␣3␣0\n");
289            d_step {
290              _count_c0 = _count_c0 + 1;
291              _minimum = _count_c0;
292              if
293                :: _count_c1 < _minimum;
294                   _minimum = _count_c1
295                :: else
296              fi;
297              if
298                :: _count_c2 < _minimum;
299                   _minimum = _count_c2
300                :: else
301              fi;
302              if
303                :: _count_s < _minimum;
304                   _minimum = _count_s
305                :: else
306              fi;
307              _count_c0 = _count_c0 - _minimum;
308              _count_c1 = _count_c1 - _minimum;
309              _count_c2 = _count_c2 - _minimum;
310              _count_s = _count_s - _minimum
311            };
312            goto st2
313          }
314     fi
315 }
316 proctype c1() {
317   if
318     :: atomic {
319          _count_c1 == 0;
320          if
321            :: s_procstates_1_0 = 1;
322               c1_st_0 = s_procstates_1_0
323          fi;
324          printf("TRANSITION␣c1␣2␣0\n");
325          d_step {
326            _count_c1 = _count_c1 + 1;
327            _minimum = _count_c0;
328            if
329              :: _count_c1 < _minimum;
330                 _minimum = _count_c1
331              :: else
332            fi;
333            if
```

```
334               :: _count_c2 < _minimum;
335                  _minimum = _count_c2
336               :: else
337             fi;
338             if
339               :: _count_s < _minimum;
340                  _minimum = _count_s
341               :: else
342             fi;
343             _count_c0 = _count_c0 - _minimum;
344             _count_c1 = _count_c1 - _minimum;
345             _count_c2 = _count_c2 - _minimum;
346             _count_s = _count_s - _minimum
347           };
348           goto st0
349         }
350       :: atomic {
351             _count_c1 == 0;
352             if
353               :: s_procstates_1_0 = 0;
354                  c1_st_0 = s_procstates_1_0
355             fi;
356             printf("TRANSITION␣c1␣2␣1\n");
357             d_step {
358               _count_c1 = _count_c1 + 1;
359               _minimum = _count_c0;
360               if
361                 :: _count_c1 < _minimum;
362                    _minimum = _count_c1
363                 :: else
364               fi;
365               if
366                 :: _count_c2 < _minimum;
367                    _minimum = _count_c2
368                 :: else
369               fi;
370               if
371                 :: _count_s < _minimum;
372                    _minimum = _count_s
373                 :: else
374               fi;
375               _count_c0 = _count_c0 - _minimum;
376               _count_c1 = _count_c1 - _minimum;
377               _count_c2 = _count_c2 - _minimum;
378               _count_s = _count_s - _minimum
379             };
380             goto st2
381           }
382     fi;
383     st0:
384       if
385         :: atomic {
386               _count_c1 == 0 && c1_proceed_0;
387               if
388                 :: s_procstates_1_0 = 2;
```

```
389                 c1_st_0 = s_procstates_1_0
390             fi;
391             printf("TRANSITION␣c1␣0␣0\n");
392             d_step {
393               _count_c1 = _count_c1 + 1;
394               _minimum = _count_c0;
395               if
396                 :: _count_c1 < _minimum;
397                    _minimum = _count_c1
398                 :: else
399               fi;
400               if
401                 :: _count_c2 < _minimum;
402                    _minimum = _count_c2
403                 :: else
404               fi;
405               if
406                 :: _count_s < _minimum;
407                    _minimum = _count_s
408                 :: else
409               fi;
410               _count_c0 = _count_c0 - _minimum;
411               _count_c1 = _count_c1 - _minimum;
412               _count_c2 = _count_c2 - _minimum;
413               _count_s = _count_s - _minimum
414             };
415             goto st1
416           }
417         :: atomic {
418             _count_c1 == 0 && !(c1_proceed_0);
419             if
420               :: s_procstates_1_0 = 1;
421                  c1_st_0 = s_procstates_1_0
422             fi;
423             printf("TRANSITION␣c1␣0␣1\n");
424             d_step {
425               _count_c1 = _count_c1 + 1;
426               _minimum = _count_c0;
427               if
428                 :: _count_c1 < _minimum;
429                    _minimum = _count_c1
430                 :: else
431               fi;
432               if
433                 :: _count_c2 < _minimum;
434                    _minimum = _count_c2
435                 :: else
436               fi;
437               if
438                 :: _count_s < _minimum;
439                    _minimum = _count_s
440                 :: else
441               fi;
442               _count_c0 = _count_c0 - _minimum;
443               _count_c1 = _count_c1 - _minimum;
```

```
444             _count_c2 = _count_c2 - _minimum;
445             _count_s = _count_s - _minimum
446           };
447           goto st0
448         }
449     fi;
450   st1:
451     if
452       :: atomic {
453           _count_c1 == 0;
454           if
455             :: s_procstates_1_0 = 3;
456                c1_st_0 = s_procstates_1_0
457           fi;
458           printf("TRANSITION c1 1 0\n");
459           d_step {
460             _count_c1 = _count_c1 + 1;
461             _minimum = _count_c0;
462             if
463               :: _count_c1 < _minimum;
464                  _minimum = _count_c1
465               :: else
466             fi;
467             if
468               :: _count_c2 < _minimum;
469                  _minimum = _count_c2
470               :: else
471             fi;
472             if
473               :: _count_s < _minimum;
474                  _minimum = _count_s
475               :: else
476             fi;
477             _count_c0 = _count_c0 - _minimum;
478             _count_c1 = _count_c1 - _minimum;
479             _count_c2 = _count_c2 - _minimum;
480             _count_s = _count_s - _minimum
481           };
482           goto st3
483         }
484       :: atomic {
485           _count_c1 == 0;
486           if
487             :: s_procstates_1_0 = 2;
488                c1_st_0 = s_procstates_1_0
489           fi;
490           printf("TRANSITION c1 1 1\n");
491           d_step {
492             _count_c1 = _count_c1 + 1;
493             _minimum = _count_c0;
494             if
495               :: _count_c1 < _minimum;
496                  _minimum = _count_c1
497               :: else
498             fi;
```

```
499             if
500               :: _count_c2 < _minimum;
501                  _minimum = _count_c2
502               :: else
503             fi;
504             if
505               :: _count_s < _minimum;
506                  _minimum = _count_s
507               :: else
508             fi;
509             _count_c0 = _count_c0 - _minimum;
510             _count_c1 = _count_c1 - _minimum;
511             _count_c2 = _count_c2 - _minimum;
512             _count_s = _count_s - _minimum
513           };
514           goto st1
515         }
516     fi;
517   st2:
518     if
519       :: atomic {
520            _count_c1 == 0;
521            if
522              :: s_procstates_1_0 = 1;
523                 c1_st_0 = s_procstates_1_0
524            fi;
525            printf("TRANSITION␣c1␣2␣0\n");
526            d_step {
527              _count_c1 = _count_c1 + 1;
528              _minimum = _count_c0;
529              if
530                :: _count_c1 < _minimum;
531                   _minimum = _count_c1
532                :: else
533              fi;
534              if
535                :: _count_c2 < _minimum;
536                   _minimum = _count_c2
537                :: else
538              fi;
539              if
540                :: _count_s < _minimum;
541                   _minimum = _count_s
542                :: else
543              fi;
544              _count_c0 = _count_c0 - _minimum;
545              _count_c1 = _count_c1 - _minimum;
546              _count_c2 = _count_c2 - _minimum;
547              _count_s = _count_s - _minimum
548            };
549            goto st0
550          }
551       :: atomic {
552            _count_c1 == 0;
553            if
```

```
554               :: s_procstates_1_0 = 0;
555                  c1_st_0 = s_procstates_1_0
556             fi;
557             printf("TRANSITION c1 2 1\n");
558             d_step {
559               _count_c1 = _count_c1 + 1;
560               _minimum = _count_c0;
561               if
562                 :: _count_c1 < _minimum;
563                    _minimum = _count_c1
564                 :: else
565               fi;
566               if
567                 :: _count_c2 < _minimum;
568                    _minimum = _count_c2
569                 :: else
570               fi;
571               if
572                 :: _count_s < _minimum;
573                    _minimum = _count_s
574                 :: else
575               fi;
576               _count_c0 = _count_c0 - _minimum;
577               _count_c1 = _count_c1 - _minimum;
578               _count_c2 = _count_c2 - _minimum;
579               _count_s = _count_s - _minimum
580             };
581             goto st2
582           }
583       fi;
584     st3:
585       if
586         :: atomic {
587              _count_c1 == 0;
588              if
589                :: s_procstates_1_0 = 0;
590                   c1_st_0 = s_procstates_1_0
591              fi;
592              printf("TRANSITION c1 3 0\n");
593              d_step {
594                _count_c1 = _count_c1 + 1;
595                _minimum = _count_c0;
596                if
597                  :: _count_c1 < _minimum;
598                     _minimum = _count_c1
599                  :: else
600                fi;
601                if
602                  :: _count_c2 < _minimum;
603                     _minimum = _count_c2
604                  :: else
605                fi;
606                if
607                  :: _count_s < _minimum;
608                     _minimum = _count_s
```

```
609                 :: else
610               fi;
611               _count_c0 = _count_c0 - _minimum;
612               _count_c1 = _count_c1 - _minimum;
613               _count_c2 = _count_c2 - _minimum;
614               _count_s = _count_s - _minimum
615             };
616             goto st2
617           }
618       fi
619   }
620   proctype c2() {
621     if
622       :: atomic {
623             _count_c2 == 0;
624             if
625               :: s_procstates_2_0 = 1;
626                  c2_st_0 = s_procstates_2_0
627             fi;
628             printf("TRANSITION␣c2␣2␣0\n");
629             d_step {
630               _count_c2 = _count_c2 + 1;
631               _minimum = _count_c0;
632               if
633                 :: _count_c1 < _minimum;
634                    _minimum = _count_c1
635                 :: else
636               fi;
637               if
638                 :: _count_c2 < _minimum;
639                    _minimum = _count_c2
640                 :: else
641               fi;
642               if
643                 :: _count_s < _minimum;
644                    _minimum = _count_s
645                 :: else
646               fi;
647               _count_c0 = _count_c0 - _minimum;
648               _count_c1 = _count_c1 - _minimum;
649               _count_c2 = _count_c2 - _minimum;
650               _count_s = _count_s - _minimum
651             };
652             goto st0
653           }
654       :: atomic {
655             _count_c2 == 0;
656             if
657               :: s_procstates_2_0 = 0;
658                  c2_st_0 = s_procstates_2_0
659             fi;
660             printf("TRANSITION␣c2␣2␣1\n");
661             d_step {
662               _count_c2 = _count_c2 + 1;
663               _minimum = _count_c0;
```

```
664             if
665               :: _count_c1 < _minimum;
666                  _minimum = _count_c1
667               :: else
668             fi;
669             if
670               :: _count_c2 < _minimum;
671                  _minimum = _count_c2
672               :: else
673             fi;
674             if
675               :: _count_s < _minimum;
676                  _minimum = _count_s
677               :: else
678             fi;
679             _count_c0 = _count_c0 - _minimum;
680             _count_c1 = _count_c1 - _minimum;
681             _count_c2 = _count_c2 - _minimum;
682             _count_s = _count_s - _minimum
683           };
684           goto st2
685         }
686     fi;
687     st0:
688       if
689         :: atomic {
690             _count_c2 == 0 && c2_proceed_0;
691             if
692               :: s_procstates_2_0 = 2;
693                  c2_st_0 = s_procstates_2_0
694             fi;
695             printf("TRANSITION␣c2␣0␣0\n");
696             d_step {
697               _count_c2 = _count_c2 + 1;
698               _minimum = _count_c0;
699               if
700                 :: _count_c1 < _minimum;
701                    _minimum = _count_c1
702                 :: else
703               fi;
704               if
705                 :: _count_c2 < _minimum;
706                    _minimum = _count_c2
707                 :: else
708               fi;
709               if
710                 :: _count_s < _minimum;
711                    _minimum = _count_s
712                 :: else
713               fi;
714               _count_c0 = _count_c0 - _minimum;
715               _count_c1 = _count_c1 - _minimum;
716               _count_c2 = _count_c2 - _minimum;
717               _count_s = _count_s - _minimum
718             };
```

```
719             goto st1
720           }
721         :: atomic {
722             _count_c2 == 0 && !(c2_proceed_0);
723             if
724               :: s_procstates_2_0 = 1;
725                  c2_st_0 = s_procstates_2_0
726             fi;
727             printf("TRANSITION␣c2␣0␣1\n");
728             d_step {
729               _count_c2 = _count_c2 + 1;
730               _minimum = _count_c0;
731               if
732                 :: _count_c1 < _minimum;
733                    _minimum = _count_c1
734                 :: else
735               fi;
736               if
737                 :: _count_c2 < _minimum;
738                    _minimum = _count_c2
739                 :: else
740               fi;
741               if
742                 :: _count_s < _minimum;
743                    _minimum = _count_s
744                 :: else
745               fi;
746               _count_c0 = _count_c0 - _minimum;
747               _count_c1 = _count_c1 - _minimum;
748               _count_c2 = _count_c2 - _minimum;
749               _count_s = _count_s - _minimum
750             };
751             goto st0
752           }
753       fi;
754     st1:
755       if
756         :: atomic {
757             _count_c2 == 0;
758             if
759               :: s_procstates_2_0 = 3;
760                  c2_st_0 = s_procstates_2_0
761             fi;
762             printf("TRANSITION␣c2␣1␣0\n");
763             d_step {
764               _count_c2 = _count_c2 + 1;
765               _minimum = _count_c0;
766               if
767                 :: _count_c1 < _minimum;
768                    _minimum = _count_c1
769                 :: else
770               fi;
771               if
772                 :: _count_c2 < _minimum;
773                    _minimum = _count_c2
```

```
774                 :: else
775               fi;
776               if
777                 :: _count_s < _minimum;
778                    _minimum = _count_s
779                 :: else
780               fi;
781               _count_c0 = _count_c0 - _minimum;
782               _count_c1 = _count_c1 - _minimum;
783               _count_c2 = _count_c2 - _minimum;
784               _count_s = _count_s - _minimum
785             };
786             goto st3
787           }
788         :: atomic {
789             _count_c2 == 0;
790             if
791               :: s_procstates_2_0 = 2;
792                  c2_st_0 = s_procstates_2_0
793             fi;
794             printf("TRANSITION c2 1 1\n");
795             d_step {
796               _count_c2 = _count_c2 + 1;
797               _minimum = _count_c0;
798               if
799                 :: _count_c1 < _minimum;
800                    _minimum = _count_c1
801                 :: else
802               fi;
803               if
804                 :: _count_c2 < _minimum;
805                    _minimum = _count_c2
806                 :: else
807               fi;
808               if
809                 :: _count_s < _minimum;
810                    _minimum = _count_s
811                 :: else
812               fi;
813               _count_c0 = _count_c0 - _minimum;
814               _count_c1 = _count_c1 - _minimum;
815               _count_c2 = _count_c2 - _minimum;
816               _count_s = _count_s - _minimum
817             };
818             goto st1
819           }
820       fi;
821     st2:
822       if
823         :: atomic {
824             _count_c2 == 0;
825             if
826               :: s_procstates_2_0 = 1;
827                  c2_st_0 = s_procstates_2_0
828             fi;
```

```
829             printf("TRANSITION␣c2␣2␣0\n");
830             d_step {
831               _count_c2 = _count_c2 + 1;
832               _minimum = _count_c0;
833               if
834                 :: _count_c1 < _minimum;
835                    _minimum = _count_c1
836                 :: else
837               fi;
838               if
839                 :: _count_c2 < _minimum;
840                    _minimum = _count_c2
841                 :: else
842               fi;
843               if
844                 :: _count_s < _minimum;
845                    _minimum = _count_s
846                 :: else
847               fi;
848               _count_c0 = _count_c0 - _minimum;
849               _count_c1 = _count_c1 - _minimum;
850               _count_c2 = _count_c2 - _minimum;
851               _count_s = _count_s - _minimum
852             };
853             goto st0
854           }
855         :: atomic {
856             _count_c2 == 0;
857             if
858               :: s_procstates_2_0 = 0;
859                  c2_st_0 = s_procstates_2_0
860             fi;
861             printf("TRANSITION␣c2␣2␣1\n");
862             d_step {
863               _count_c2 = _count_c2 + 1;
864               _minimum = _count_c0;
865               if
866                 :: _count_c1 < _minimum;
867                    _minimum = _count_c1
868                 :: else
869               fi;
870               if
871                 :: _count_c2 < _minimum;
872                    _minimum = _count_c2
873                 :: else
874               fi;
875               if
876                 :: _count_s < _minimum;
877                    _minimum = _count_s
878                 :: else
879               fi;
880               _count_c0 = _count_c0 - _minimum;
881               _count_c1 = _count_c1 - _minimum;
882               _count_c2 = _count_c2 - _minimum;
883               _count_s = _count_s - _minimum
```

```
884             };
885             goto st2
886           }
887       fi;
888     st3:
889       if
890         :: atomic {
891             _count_c2 == 0;
892             if
893               :: s_procstates_2_0 = 0;
894                  c2_st_0 = s_procstates_2_0
895             fi;
896             printf("TRANSITION␣c2␣3␣0\n");
897             d_step {
898               _count_c2 = _count_c2 + 1;
899               _minimum = _count_c0;
900               if
901                 :: _count_c1 < _minimum;
902                    _minimum = _count_c1
903                 :: else
904               fi;
905               if
906                 :: _count_c2 < _minimum;
907                    _minimum = _count_c2
908                 :: else
909               fi;
910               if
911                 :: _count_s < _minimum;
912                    _minimum = _count_s
913                 :: else
914               fi;
915               _count_c0 = _count_c0 - _minimum;
916               _count_c1 = _count_c1 - _minimum;
917               _count_c2 = _count_c2 - _minimum;
918               _count_s = _count_s - _minimum
919             };
920             goto st2
921           }
922       fi
923   }
924   proctype s() {
925     if
926       :: atomic {
927           _count_s == 0 && s_procstates_0_0 != 1 && s_procstates_1_0 != 1 && ↩
      ↪s_procstates_2_0 == 1 && s_procstates_0_0 != 2 && s_procstates_1_0 != 2;
928           c0_proceed_0 = false;
929           c1_proceed_0 = false;
930           c2_proceed_0 = true;
931           printf("TRANSITION␣s␣0␣0\n");
932           d_step {
933             _count_s = _count_s + 1;
934             _minimum = _count_c0;
935             if
936               :: _count_c1 < _minimum;
937                  _minimum = _count_c1
```

```
938             :: else
939           fi;
940           if
941             :: _count_c2 < _minimum;
942                _minimum = _count_c2
943             :: else
944           fi;
945           if
946             :: _count_s < _minimum;
947                _minimum = _count_s
948             :: else
949           fi;
950           _count_c0 = _count_c0 - _minimum;
951           _count_c1 = _count_c1 - _minimum;
952           _count_c2 = _count_c2 - _minimum;
953           _count_s = _count_s - _minimum
954         };
955         goto st0
956       }
957     :: atomic {
958         _count_s == 0 && s_procstates_0_0 == 1 && s_procstates_1_0 != 2 && ↩
    ↪s_procstates_2_0 != 2;
959         c0_proceed_0 = true;
960         c1_proceed_0 = false;
961         c2_proceed_0 = false;
962         printf("TRANSITION␣s␣0␣1\n");
963         d_step {
964           _count_s = _count_s + 1;
965           _minimum = _count_c0;
966           if
967             :: _count_c1 < _minimum;
968                _minimum = _count_c1
969             :: else
970           fi;
971           if
972             :: _count_c2 < _minimum;
973                _minimum = _count_c2
974             :: else
975           fi;
976           if
977             :: _count_s < _minimum;
978                _minimum = _count_s
979             :: else
980           fi;
981           _count_c0 = _count_c0 - _minimum;
982           _count_c1 = _count_c1 - _minimum;
983           _count_c2 = _count_c2 - _minimum;
984           _count_s = _count_s - _minimum
985         };
986         goto st0
987       }
988     :: atomic {
989         _count_s == 0;
990         c0_proceed_0 = false;
991         c1_proceed_0 = false;
```

```
992           c2_proceed_0 = false;
993           printf("TRANSITION␣s␣0␣2\n");
994           d_step {
995             _count_s = _count_s + 1;
996             _minimum = _count_c0;
997             if
998               :: _count_c1 < _minimum;
999                  _minimum = _count_c1
1000              :: else
1001            fi;
1002            if
1003              :: _count_c2 < _minimum;
1004                 _minimum = _count_c2
1005              :: else
1006            fi;
1007            if
1008              :: _count_s < _minimum;
1009                 _minimum = _count_s
1010              :: else
1011            fi;
1012            _count_c0 = _count_c0 - _minimum;
1013            _count_c1 = _count_c1 - _minimum;
1014            _count_c2 = _count_c2 - _minimum;
1015            _count_s = _count_s - _minimum
1016          };
1017          goto st0
1018        }
1019     :: atomic {
1020          _count_s == 0 && s_procstates_0_0 != 1 && s_procstates_1_0 == 1 && ↩
        ↪s_procstates_0_0 != 2 && s_procstates_2_0 != 2;
1021          c0_proceed_0 = false;
1022          c1_proceed_0 = true;
1023          c2_proceed_0 = false;
1024          printf("TRANSITION␣s␣0␣3\n");
1025          d_step {
1026            _count_s = _count_s + 1;
1027            _minimum = _count_c0;
1028            if
1029              :: _count_c1 < _minimum;
1030                 _minimum = _count_c1
1031              :: else
1032            fi;
1033            if
1034              :: _count_c2 < _minimum;
1035                 _minimum = _count_c2
1036              :: else
1037            fi;
1038            if
1039              :: _count_s < _minimum;
1040                 _minimum = _count_s
1041              :: else
1042            fi;
1043            _count_c0 = _count_c0 - _minimum;
1044            _count_c1 = _count_c1 - _minimum;
1045            _count_c2 = _count_c2 - _minimum;
```

```
1046            _count_s = _count_s - _minimum
1047          };
1048          goto st0
1049        }
1050    fi;
1051    st0:
1052      if
1053        :: atomic {
1054            _count_s == 0 && s_procstates_0_0 != 1 && s_procstates_1_0 != 1 && ↩
     ↪s_procstates_2_0 == 1 && s_procstates_0_0 != 2 && s_procstates_1_0 != 2;
1055            c0_proceed_0 = false;
1056            c1_proceed_0 = false;
1057            c2_proceed_0 = true;
1058            printf("TRANSITION␣s␣0␣0\n");
1059            d_step {
1060              _count_s = _count_s + 1;
1061              _minimum = _count_c0;
1062              if
1063                :: _count_c1 < _minimum;
1064                   _minimum = _count_c1
1065                :: else
1066              fi;
1067              if
1068                :: _count_c2 < _minimum;
1069                   _minimum = _count_c2
1070                :: else
1071              fi;
1072              if
1073                :: _count_s < _minimum;
1074                   _minimum = _count_s
1075                :: else
1076              fi;
1077              _count_c0 = _count_c0 - _minimum;
1078              _count_c1 = _count_c1 - _minimum;
1079              _count_c2 = _count_c2 - _minimum;
1080              _count_s = _count_s - _minimum
1081            };
1082            goto st0
1083          }
1084        :: atomic {
1085            _count_s == 0 && s_procstates_0_0 == 1 && s_procstates_1_0 != 2 && ↩
     ↪s_procstates_2_0 != 2;
1086            c0_proceed_0 = true;
1087            c1_proceed_0 = false;
1088            c2_proceed_0 = false;
1089            printf("TRANSITION␣s␣0␣1\n");
1090            d_step {
1091              _count_s = _count_s + 1;
1092              _minimum = _count_c0;
1093              if
1094                :: _count_c1 < _minimum;
1095                   _minimum = _count_c1
1096                :: else
1097              fi;
1098              if
```

```
1099                :: _count_c2 < _minimum;
1100                   _minimum = _count_c2
1101                :: else
1102              fi;
1103              if
1104                :: _count_s < _minimum;
1105                   _minimum = _count_s
1106                :: else
1107              fi;
1108              _count_c0 = _count_c0 - _minimum;
1109              _count_c1 = _count_c1 - _minimum;
1110              _count_c2 = _count_c2 - _minimum;
1111              _count_s = _count_s - _minimum
1112            };
1113            goto st0
1114          }
1115        :: atomic {
1116            _count_s == 0;
1117            c0_proceed_0 = false;
1118            c1_proceed_0 = false;
1119            c2_proceed_0 = false;
1120            printf("TRANSITION␣s␣0␣2\n");
1121            d_step {
1122              _count_s = _count_s + 1;
1123              _minimum = _count_c0;
1124              if
1125                :: _count_c1 < _minimum;
1126                   _minimum = _count_c1
1127                :: else
1128              fi;
1129              if
1130                :: _count_c2 < _minimum;
1131                   _minimum = _count_c2
1132                :: else
1133              fi;
1134              if
1135                :: _count_s < _minimum;
1136                   _minimum = _count_s
1137                :: else
1138              fi;
1139              _count_c0 = _count_c0 - _minimum;
1140              _count_c1 = _count_c1 - _minimum;
1141              _count_c2 = _count_c2 - _minimum;
1142              _count_s = _count_s - _minimum
1143            };
1144            goto st0
1145          }
1146        :: atomic {
1147            _count_s == 0 && s_procstates_0_0 != 1 && s_procstates_1_0 == 1 && ↩
    ↪s_procstates_0_0 != 2 && s_procstates_2_0 != 2;
1148            c0_proceed_0 = false;
1149            c1_proceed_0 = true;
1150            c2_proceed_0 = false;
1151            printf("TRANSITION␣s␣0␣3\n");
1152            d_step {
```

```
1153                _count_s = _count_s + 1;
1154                _minimum = _count_c0;
1155                if
1156                  :: _count_c1 < _minimum;
1157                     _minimum = _count_c1
1158                  :: else
1159                fi;
1160                if
1161                  :: _count_c2 < _minimum;
1162                     _minimum = _count_c2
1163                  :: else
1164                fi;
1165                if
1166                  :: _count_s < _minimum;
1167                     _minimum = _count_s
1168                  :: else
1169                fi;
1170                _count_c0 = _count_c0 - _minimum;
1171                _count_c1 = _count_c1 - _minimum;
1172                _count_c2 = _count_c2 - _minimum;
1173                _count_s = _count_s - _minimum
1174              };
1175              goto st0
1176            }
1177      fi
1178  }
1179  init {
1180    atomic {
1181      skip;
1182      run c0();
1183      run c1();
1184      run c2();
1185      run s()
1186    }
1187  }
1188  never {
1189    if
1190      :: atomic {
1191           c2_st_0 == 2 && c0_st_0 == 2;
1192           goto st3
1193         }
1194      :: atomic {
1195           goto st0
1196         }
1197      :: atomic {
1198           c1_st_0 == 2 && c0_st_0 == 2;
1199           goto st3
1200         }
1201      :: atomic {
1202           c2_st_0 == 2 && c1_st_0 == 2;
1203           goto st3
1204         }
1205      :: atomic {
1206           goto st1
1207         }
```

```
1208      :: atomic {
1209           c0_st_0 == 2 && c1_st_0 == 2;
1210           goto st3
1211         }
1212      :: atomic {
1213           c1_st_0 == 2 && c2_st_0 == 2;
1214           goto st3
1215         }
1216      :: atomic {
1217           goto st2
1218         }
1219      :: atomic {
1220           c0_st_0 == 2 && c2_st_0 == 2;
1221           goto st3
1222         }
1223    fi;
1224    st0:
1225      if
1226        :: atomic {
1227             c2_st_0 == 2 && c0_st_0 == 2;
1228             goto st3
1229           }
1230        :: atomic {
1231             goto st0
1232           }
1233        :: atomic {
1234             c1_st_0 == 2 && c0_st_0 == 2;
1235             goto st3
1236           }
1237      fi;
1238    st1:
1239      if
1240        :: atomic {
1241             c2_st_0 == 2 && c1_st_0 == 2;
1242             goto st3
1243           }
1244        :: atomic {
1245             goto st1
1246           }
1247        :: atomic {
1248             c0_st_0 == 2 && c1_st_0 == 2;
1249             goto st3
1250           }
1251      fi;
1252    st2:
1253      if
1254        :: atomic {
1255             c1_st_0 == 2 && c2_st_0 == 2;
1256             goto st3
1257           }
1258        :: atomic {
1259             goto st2
1260           }
1261        :: atomic {
1262             c0_st_0 == 2 && c2_st_0 == 2;
```

```
1263              goto st3
1264            }
1265      fi;
1266    accept3:
1267      st3:
1268        if
1269          :: atomic {
1270               goto st3
1271             }
1272        fi
1273  }
```

B.3 UPPAAL XML Files Generated by the GTL-Tool

The following UPPAAL-files were generated by GTL-0.1 (2012-01-09) via processing of `mutex_<N>.glt` (see B.1.1ff):

```
gtl -m uppaal mutex_<N>.gtl
```

The corresponding queries `mutex_<N>.q` were generated by the script `gen-mutex-gtl.awk` (see A.1).

B.3.1 UPPAAL Model: Mutex with 2 Clients

FILE: `mutex_2.xml`

```
1  <nta>
2    <declaration>int[0,1] c0_proceed[1] = {0};
3  int[0,3] c0_st[1] = {0};
4  int[0,1] c1_proceed[1] = {0};
5  int[0,3] c1_st[1] = {0};
6  int[0,3] s_procstates_0[1] = {0};
7  int[0,3] s_procstates_1[1] = {0};</declaration>
8    <template>
9      <name>c0_tmpl</name>
10     <location id="start">
11       <name>start</name>
12       <urgent/>
13     </location>
14     <location id="l0">
15       <name>l0</name>
16     </location>
17     <location id="l1">
18       <name>l1</name>
19     </location>
20     <location id="l2">
21       <name>l2</name>
22     </location>
23     <location id="l3">
24       <name>l3</name>
25     </location>
26     <init ref="start"/>
```

```
27      <transition>
28        <source ref="start"/>
29        <target ref="l0"/>
30        <label kind="assignment">s_procstates_0[0] = 1, c0_st[0] = 1</label>
31      </transition>
32      <transition>
33        <source ref="start"/>
34        <target ref="l2"/>
35        <label kind="assignment">s_procstates_0[0] = 0, c0_st[0] = 0</label>
36      </transition>
37      <transition>
38        <source ref="l0"/>
39        <target ref="l1"/>
40        <label kind="guard">c0_proceed[0]</label>
41        <label kind="assignment">s_procstates_0[0] = 2, c0_st[0] = 2</label>
42      </transition>
43      <transition>
44        <source ref="l0"/>
45        <target ref="l0"/>
46        <label kind="guard">!c0_proceed[0]</label>
47        <label kind="assignment">s_procstates_0[0] = 1, c0_st[0] = 1</label>
48      </transition>
49      <transition>
50        <source ref="l1"/>
51        <target ref="l3"/>
52        <label kind="assignment">s_procstates_0[0] = 3, c0_st[0] = 3</label>
53      </transition>
54      <transition>
55        <source ref="l1"/>
56        <target ref="l1"/>
57        <label kind="assignment">s_procstates_0[0] = 2, c0_st[0] = 2</label>
58      </transition>
59      <transition>
60        <source ref="l2"/>
61        <target ref="l0"/>
62        <label kind="assignment">s_procstates_0[0] = 1, c0_st[0] = 1</label>
63      </transition>
64      <transition>
65        <source ref="l2"/>
66        <target ref="l2"/>
67        <label kind="assignment">s_procstates_0[0] = 0, c0_st[0] = 0</label>
68      </transition>
69      <transition>
70        <source ref="l3"/>
71        <target ref="l2"/>
72        <label kind="assignment">s_procstates_0[0] = 0, c0_st[0] = 0</label>
73      </transition>
74    </template>
75    <template>
76      <name>c1_tmpl</name>
77      <location id="start">
78        <name>start</name>
79        <urgent/>
80      </location>
81      <location id="l0">
```

```
82        <name>l0</name>
83      </location>
84      <location id="l1">
85        <name>l1</name>
86      </location>
87      <location id="l2">
88        <name>l2</name>
89      </location>
90      <location id="l3">
91        <name>l3</name>
92      </location>
93      <init ref="start"/>
94      <transition>
95        <source ref="start"/>
96        <target ref="l0"/>
97        <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
98      </transition>
99      <transition>
100       <source ref="start"/>
101       <target ref="l2"/>
102       <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
103     </transition>
104     <transition>
105       <source ref="l0"/>
106       <target ref="l1"/>
107       <label kind="guard">c1_proceed[0]</label>
108       <label kind="assignment">s_procstates_1[0] = 2, c1_st[0] = 2</label>
109     </transition>
110     <transition>
111       <source ref="l0"/>
112       <target ref="l0"/>
113       <label kind="guard">!c1_proceed[0]</label>
114       <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
115     </transition>
116     <transition>
117       <source ref="l1"/>
118       <target ref="l3"/>
119       <label kind="assignment">s_procstates_1[0] = 3, c1_st[0] = 3</label>
120     </transition>
121     <transition>
122       <source ref="l1"/>
123       <target ref="l1"/>
124       <label kind="assignment">s_procstates_1[0] = 2, c1_st[0] = 2</label>
125     </transition>
126     <transition>
127       <source ref="l2"/>
128       <target ref="l0"/>
129       <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
130     </transition>
131     <transition>
132       <source ref="l2"/>
133       <target ref="l2"/>
134       <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
135     </transition>
136     <transition>
```

```
137       <source ref="l3"/>
138       <target ref="l2"/>
139       <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
140     </transition>
141   </template>
142   <template>
143     <name>s_tmpl</name>
144     <location id="start">
145       <name>start</name>
146       <urgent/>
147     </location>
148     <location id="l0">
149       <name>l0</name>
150     </location>
151     <init ref="start"/>
152     <transition>
153       <source ref="start"/>
154       <target ref="l0"/>
155       <label kind="guard">s_procstates_0[0] != 1 and s_procstates_1[0] == 1 and ↩
    ↪s_procstates_0[0] != 2</label>
156       <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 1</label>
157     </transition>
158     <transition>
159       <source ref="start"/>
160       <target ref="l0"/>
161       <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0</label>
162     </transition>
163     <transition>
164       <source ref="start"/>
165       <target ref="l0"/>
166       <label kind="guard">s_procstates_0[0] == 1 and s_procstates_1[0] != 2</↩
    ↪label>
167       <label kind="assignment">c0_proceed[0] = 1, c1_proceed[0] = 0</label>
168     </transition>
169     <transition>
170       <source ref="l0"/>
171       <target ref="l0"/>
172       <label kind="guard">s_procstates_0[0] != 1 and s_procstates_1[0] == 1 and ↩
    ↪s_procstates_0[0] != 2</label>
173       <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 1</label>
174     </transition>
175     <transition>
176       <source ref="l0"/>
177       <target ref="l0"/>
178       <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0</label>
179     </transition>
180     <transition>
181       <source ref="l0"/>
182       <target ref="l0"/>
183       <label kind="guard">s_procstates_0[0] == 1 and s_procstates_1[0] != 2</↩
    ↪label>
184       <label kind="assignment">c0_proceed[0] = 1, c1_proceed[0] = 0</label>
185     </transition>
186   </template>
187   <system>c0 = c0_tmpl();
```

```
c1 = c1_tmpl();
s = s_tmpl();
system c0, c1, s;</system>
</nta>
```

FILE: mutex_2.q

```
// GENERATED BY gen-mutex-gtl.awk
A[] (not (false || (c0.l1 && c1.l1)  ))
```

B.3.2 UPPAAL Model: Mutex with 3 Clients

FILE: mutex_3.xml

```
<nta>
  <declaration>int[0,1] c0_proceed[1] = {0};
int[0,3] c0_st[1] = {0};
int[0,1] c1_proceed[1] = {0};
int[0,3] c1_st[1] = {0};
int[0,1] c2_proceed[1] = {0};
int[0,3] c2_st[1] = {0};
int[0,3] s_procstates_0[1] = {0};
int[0,3] s_procstates_1[1] = {0};
int[0,3] s_procstates_2[1] = {0};</declaration>
  <template>
    <name>c0_tmpl</name>
    <location id="start">
      <name>start</name>
      <urgent/>
    </location>
    <location id="l0">
      <name>l0</name>
    </location>
    <location id="l1">
      <name>l1</name>
    </location>
    <location id="l2">
      <name>l2</name>
    </location>
    <location id="l3">
      <name>l3</name>
    </location>
    <init ref="start"/>
    <transition>
      <source ref="start"/>
      <target ref="l0"/>
      <label kind="assignment">s_procstates_0[0] = 1, c0_st[0] = 1</label>
    </transition>
    <transition>
      <source ref="start"/>
      <target ref="l2"/>
      <label kind="assignment">s_procstates_0[0] = 0, c0_st[0] = 0</label>
    </transition>
```

```
40      <transition>
41        <source ref="l0"/>
42        <target ref="l1"/>
43        <label kind="guard">c0_proceed[0]</label>
44        <label kind="assignment">s_procstates_0[0] = 2, c0_st[0] = 2</label>
45      </transition>
46      <transition>
47        <source ref="l0"/>
48        <target ref="l0"/>
49        <label kind="guard">!c0_proceed[0]</label>
50        <label kind="assignment">s_procstates_0[0] = 1, c0_st[0] = 1</label>
51      </transition>
52      <transition>
53        <source ref="l1"/>
54        <target ref="l3"/>
55        <label kind="assignment">s_procstates_0[0] = 3, c0_st[0] = 3</label>
56      </transition>
57      <transition>
58        <source ref="l1"/>
59        <target ref="l1"/>
60        <label kind="assignment">s_procstates_0[0] = 2, c0_st[0] = 2</label>
61      </transition>
62      <transition>
63        <source ref="l2"/>
64        <target ref="l0"/>
65        <label kind="assignment">s_procstates_0[0] = 1, c0_st[0] = 1</label>
66      </transition>
67      <transition>
68        <source ref="l2"/>
69        <target ref="l2"/>
70        <label kind="assignment">s_procstates_0[0] = 0, c0_st[0] = 0</label>
71      </transition>
72      <transition>
73        <source ref="l3"/>
74        <target ref="l2"/>
75        <label kind="assignment">s_procstates_0[0] = 0, c0_st[0] = 0</label>
76      </transition>
77    </template>
78    <template>
79      <name>c1_tmpl</name>
80      <location id="start">
81        <name>start</name>
82        <urgent/>
83      </location>
84      <location id="l0">
85        <name>l0</name>
86      </location>
87      <location id="l1">
88        <name>l1</name>
89      </location>
90      <location id="l2">
91        <name>l2</name>
92      </location>
93      <location id="l3">
94        <name>l3</name>
```

```
95      </location>
96      <init ref="start"/>
97      <transition>
98        <source ref="start"/>
99        <target ref="l0"/>
100       <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
101     </transition>
102     <transition>
103       <source ref="start"/>
104       <target ref="l2"/>
105       <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
106     </transition>
107     <transition>
108       <source ref="l0"/>
109       <target ref="l1"/>
110       <label kind="guard">c1_proceed[0]</label>
111       <label kind="assignment">s_procstates_1[0] = 2, c1_st[0] = 2</label>
112     </transition>
113     <transition>
114       <source ref="l0"/>
115       <target ref="l0"/>
116       <label kind="guard">!c1_proceed[0]</label>
117       <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
118     </transition>
119     <transition>
120       <source ref="l1"/>
121       <target ref="l3"/>
122       <label kind="assignment">s_procstates_1[0] = 3, c1_st[0] = 3</label>
123     </transition>
124     <transition>
125       <source ref="l1"/>
126       <target ref="l1"/>
127       <label kind="assignment">s_procstates_1[0] = 2, c1_st[0] = 2</label>
128     </transition>
129     <transition>
130       <source ref="l2"/>
131       <target ref="l0"/>
132       <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
133     </transition>
134     <transition>
135       <source ref="l2"/>
136       <target ref="l2"/>
137       <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
138     </transition>
139     <transition>
140       <source ref="l3"/>
141       <target ref="l2"/>
142       <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
143     </transition>
144   </template>
145   <template>
146     <name>c2_tmpl</name>
147     <location id="start">
148       <name>start</name>
149       <urgent/>
```

```
150     </location>
151     <location id="l0">
152       <name>l0</name>
153     </location>
154     <location id="l1">
155       <name>l1</name>
156     </location>
157     <location id="l2">
158       <name>l2</name>
159     </location>
160     <location id="l3">
161       <name>l3</name>
162     </location>
163     <init ref="start"/>
164     <transition>
165       <source ref="start"/>
166       <target ref="l0"/>
167       <label kind="assignment">s_procstates_2[0] = 1, c2_st[0] = 1</label>
168     </transition>
169     <transition>
170       <source ref="start"/>
171       <target ref="l2"/>
172       <label kind="assignment">s_procstates_2[0] = 0, c2_st[0] = 0</label>
173     </transition>
174     <transition>
175       <source ref="l0"/>
176       <target ref="l1"/>
177       <label kind="guard">c2_proceed[0]</label>
178       <label kind="assignment">s_procstates_2[0] = 2, c2_st[0] = 2</label>
179     </transition>
180     <transition>
181       <source ref="l0"/>
182       <target ref="l0"/>
183       <label kind="guard">!c2_proceed[0]</label>
184       <label kind="assignment">s_procstates_2[0] = 1, c2_st[0] = 1</label>
185     </transition>
186     <transition>
187       <source ref="l1"/>
188       <target ref="l3"/>
189       <label kind="assignment">s_procstates_2[0] = 3, c2_st[0] = 3</label>
190     </transition>
191     <transition>
192       <source ref="l1"/>
193       <target ref="l1"/>
194       <label kind="assignment">s_procstates_2[0] = 2, c2_st[0] = 2</label>
195     </transition>
196     <transition>
197       <source ref="l2"/>
198       <target ref="l0"/>
199       <label kind="assignment">s_procstates_2[0] = 1, c2_st[0] = 1</label>
200     </transition>
201     <transition>
202       <source ref="l2"/>
203       <target ref="l2"/>
204       <label kind="assignment">s_procstates_2[0] = 0, c2_st[0] = 0</label>
```

```
205     </transition>
206     <transition>
207       <source ref="l3"/>
208       <target ref="l2"/>
209       <label kind="assignment">s_procstates_2[0] = 0, c2_st[0] = 0</label>
210     </transition>
211   </template>
212   <template>
213     <name>s_tmpl</name>
214     <location id="start">
215       <name>start</name>
216       <urgent/>
217     </location>
218     <location id="l0">
219       <name>l0</name>
220     </location>
221     <init ref="start"/>
222     <transition>
223       <source ref="start"/>
224       <target ref="l0"/>
225       <label kind="guard">s_procstates_0[0] != 1 and s_procstates_1[0] != 1 and ↩
    ↪s_procstates_2[0] == 1 and s_procstates_0[0] != 2 and s_procstates_1[0] != ↩
    ↪2</label>
226       <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0, c2_proceed↩
    ↪[0] = 1</label>
227     </transition>
228     <transition>
229       <source ref="start"/>
230       <target ref="l0"/>
231       <label kind="guard">s_procstates_0[0] == 1 and s_procstates_1[0] != 2 and ↩
    ↪s_procstates_2[0] != 2</label>
232       <label kind="assignment">c0_proceed[0] = 1, c1_proceed[0] = 0, c2_proceed↩
    ↪[0] = 0</label>
233     </transition>
234     <transition>
235       <source ref="start"/>
236       <target ref="l0"/>
237       <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0, c2_proceed↩
    ↪[0] = 0</label>
238     </transition>
239     <transition>
240       <source ref="start"/>
241       <target ref="l0"/>
242       <label kind="guard">s_procstates_0[0] != 1 and s_procstates_1[0] == 1 and ↩
    ↪s_procstates_0[0] != 2 and s_procstates_2[0] != 2</label>
243       <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 1, c2_proceed↩
    ↪[0] = 0</label>
244     </transition>
245     <transition>
246       <source ref="l0"/>
247       <target ref="l0"/>
248       <label kind="guard">s_procstates_0[0] != 1 and s_procstates_1[0] != 1 and ↩
    ↪s_procstates_2[0] == 1 and s_procstates_0[0] != 2 and s_procstates_1[0] != ↩
    ↪2</label>
```

```
249       <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0, c2_proceed[0] = 1</label>
250      </transition>
251      <transition>
252        <source ref="l0"/>
253        <target ref="l0"/>
254        <label kind="guard">s_procstates_0[0] == 1 and s_procstates_1[0] != 2 and s_procstates_2[0] != 2</label>
255        <label kind="assignment">c0_proceed[0] = 1, c1_proceed[0] = 0, c2_proceed[0] = 0</label>
256      </transition>
257      <transition>
258        <source ref="l0"/>
259        <target ref="l0"/>
260        <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0, c2_proceed[0] = 0</label>
261      </transition>
262      <transition>
263        <source ref="l0"/>
264        <target ref="l0"/>
265        <label kind="guard">s_procstates_0[0] != 1 and s_procstates_1[0] == 1 and s_procstates_0[0] != 2 and s_procstates_2[0] != 2</label>
266        <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 1, c2_proceed[0] = 0</label>
267      </transition>
268    </template>
269    <system>c0 = c0_tmpl();
270 c1 = c1_tmpl();
271 c2 = c2_tmpl();
272 s = s_tmpl();
273 system c0, c1, c2, s;</system>
274 </nta>
```

FILE: mutex_3.q

```
1 // GENERATED BY gen-mutex-gtl.awk
2 A[] (not (false || (c0.l1 && c1.l1) || (c0.l1 && c2.l1) || (c1.l1 && c2.l1)  ))
```

Appendix C. Tool Information

C.1 GTL tool

Version: 0.1 (2012-01-09)
Output of `gtl --help`

```
Usage: gtl [OPTION...] gtl-file
Used environment variables:
 * CC - Path to compiler
 * CFLAGS - Additional flags to be passed to compiler
 * LDFLAGS - Additional flags to be passed to linker
 * SCADE_ROOT - Path to the Scade root directory (e.g. C:\Program Files\Esterel Technologies\SCADE 6.1.2)
 All environment variables may be passed in the form <Variable>=<Value> as option.

 -m mode              --mode=mode                    The tranlation mode ("native-c", "local", "promela-buddy", "pretty", "native"(default) or "uppaal")
 -t file              --trace-file=file              Use a trace file to restrict a simulation
 -o path              --output-directory=path        Path into which the output should be generated
 -h                   --help                         Show this help information
 -v                   --version                      Show version information
 -V[verbosity level]  --verbosity[=verbosity level] How much additional information is printed? (default 1)
 -n                   --dry-run                      Perform a dry run only generating files and not executing anything.
 -d option            --debug=option                 Give debugging options (e.g. -ddump-buchi)
```

C.2 SPIN tool

Version: 6.1.0
Output of (generated) verifier `mutex_*.verifier --help`

```
saw option --
Spin Version 6.1.0 -- 4 May 2011
Valid Options are:
    -a  find acceptance cycles
    -A  ignore assert() violations
    -b  consider it an error to exceed the depth-limit
    -cN stop at Nth error (defaults to -c1)
    -D  print state tables in dot-format and stop
    -d  print state tables and stop
```

```
    -e  create trails for all errors
    -E  ignore invalid end states
    -f  add weak fairness (to -a or -l)
    -hN use different hash-seed N:1..32
    -i  search for shortest path to error
    -I  like -i, but approximate and faster
    -J  reverse eval order of nested unlesses
    -l  find non-progress cycles -> disabled, requires compilation with -DNP
    -mN max depth N steps (default=10k)
    -n  no listing of unreached states
    -QN set time-limit on execution of N minutes
    -q  require empty chans in valid end states
    -r  read and execute trail - can add -v,-n,-PN,-g,-C
    -rN read and execute N-th error trail
    -C  read and execute trail - columnated output (can add -v,-n)
    -PN read and execute trail - restrict trail output to proc N
    -g  read and execute trail + msc gui support
    -S  silent replay: only user defined printfs show
    -T  create trail files in read-only mode
    -tsuf replace .trail with .suf on trailfiles
    -V  print SPIN version number
    -v  verbose -- filenames in unreached state listing
    -wN hashtable of 2^N entries (defaults to -w19)
    -x  do not overwrite an existing trail file

    options -r, -C, -PN, -g, and -S can optionally be followed by
    a filename argument, as in '-r filename', naming the trailfile
```

Output of (generated) verifier `mutex_*.verifier --help` **when compiled with** `-DSAFETY`

```
saw option --
Spin Version 6.1.0 -- 4 May 2011
Valid Options are:
    -a,-l,-f  -> are disabled by -DSAFETY
    -A  ignore assert() violations
    -b  consider it an error to exceed the depth-limit
    -cN stop at Nth error (defaults to -c1)
    -D  print state tables in dot-format and stop
    -d  print state tables and stop
    -e  create trails for all errors
    -E  ignore invalid end states
    -hN use different hash-seed N:1..32
    -i  search for shortest path to error
    -I  like -i, but approximate and faster
    -J  reverse eval order of nested unlesses
    -mN max depth N steps (default=10k)
    -n  no listing of unreached states
    -QN set time-limit on execution of N minutes
    -q  require empty chans in valid end states
    -r  read and execute trail - can add -v,-n,-PN,-g,-C
    -rN read and execute N-th error trail
    -C  read and execute trail - columnated output (can add -v,-n)
    -PN read and execute trail - restrict trail output to proc N
    -g  read and execute trail + msc gui support
```

```
    -S  silent replay: only user defined printfs show
    -T  create trail files in read-only mode
    -tsuf replace .trail with .suf on trailfiles
    -V  print SPIN version number
    -v  verbose -- filenames in unreached state listing
    -wN hashtable of 2^N entries (defaults to -w19)
    -x  do not overwrite an existing trail file

    options -r, -C, -PN, -g, and -S can optionally be followed by
    a filename argument, as in '-r filename', naming the trailfile
```

C.2.1 C Compiler used to Compile SPIN Verifiers

Output of `gcc --version`

```
[oli@bull Mutex-Benchmark]$ gcc --version
gcc (GCC) 4.1.2 20080704 (Red Hat 4.1.2-51)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

C.3 UPPAAL tool

Version: 4.1.7
Output of `verifyta --help`

```
Usage: verifyta [OPTION]... MODEL QUERY
where MODEL is a model file and QUERY is a query file.
If QUERY is missing it will be guessed.

Tuning options:
  -A  Use convex-hull approximation.
  -C  Disable most memory reduction techniques.
  -H n
      Set hash table size for bit state hashing to 2**n
      (default = 27)
  -n <0|1|2|3|4>
      Select extrapolation operator.
        0: Automatic
        1: No extrapolation (use with care)
        2: Difference extrapolation
        3: Location based extrapolation
        4: Lower/upper extrapolation
  -o <0|1|2|3|4>
      Select search order.
        0: Breadth first (default, -b too)
        1: Depth first (-d too)
        2: Random depth first
        3: Optimal first (requires -t1 or -t2)
        4: Random optimal depth first (requires -t1 or -t2)
        6: Target first
  -S <0|1|2>
```

```
      Optimize space consumption (0 = none, 1 = default, 2 = most)
  -T  Reuse state space when several properties are examined.
  -Z  Use bit-state hashing.

Options for trace generation:
  -f prefix
      Write symbolic traces to files 'prefix-n.xtr' rather than to stderr.
  -t <0|1|2>
      Generate diagnostic information on stderr.
        0: Some trace
        1: Shortest trace (disables reuse)
        2: Fastest trace (disables reuse)
  -y  Display traces symbolically (post-stable).
  -Y  Display traces symbolically (pre- and post-stable).

Miscellaneous options:
  -h  Shows this help screen.
  -q  Do not display the option summary.
  -r seed
      Set seed for random number generator (default is current time).
  -s  Do not display the progress indicator.
  -u  Show summary after verification (incorrect for liveness properties).
  -v  Show version number.
  -x filename
      Save the (modified) system in XML format, in file 'filename'.
      If the system has been modified, an associated query file is created.
  -X prefix
      Save the symbolic trace in file 'prefixPropertynumber.xml'
  -p double
      lower (delta) probabilistic uncertainty.
  -P double
      upper (delta) probabilistic uncertainty.
  -a double
      probability of false negatives (alpha).
  -B double
      probability of false positives (beta).
  -E double
      probability uncertainty (epsilon).
  -i double
      threshold u0 for comparing probabilities.
  -j double
      threshold u1 for comparing probabilities.
  -w double
      histogram bar width.
  -c 0|1
      activate (1) or not (0) the SMC parametric comparison.
  -V activate SMC coverage.
  -M activate Modest semantics for updates.
  -L filename
      store the vector with the results of all simulations for quantitative SMC.

The ordering of options is significant.

Environment variables:
  UPPAAL_DISABLE_SWEEPLINE    disable sweepline method
```

```
82    UPPAAL_DISABLE_OPTIMISER    disable peephole optimiser
83    UPPAAL_DISABLE_SYMMETRY     disable symmetry reduction
84    UPPAAL_COMPILE_ONLY         write compiled model to stdout and stop
85    UPPAAL_OLD_SYNTAX           use version 3.4 syntax
86
87  The value of these variables does not matter. Defining them is
88  enough to activate the feature in question.
```

Appendix D. Platform Information

FILE: brief_info_bull.txt

```
[oli@bull]$ uname -a
Linux bull 2.6.18-274.3.1.el5 #1 SMP Tue Sep 6 20:13:52 EDT 2011 x86_64 x86_64 x86_64 GNU/Linux
[oli@bull]$ ll /etc/*-release
-rw-r--r-- 1 root root 27 Aug 29 13:00 /etc/redhat-release
[oli@bull]$ cat /etc/redhat-release
CentOS release 5.7 (Final)
[oli@bull]$ date
Fri Feb  3 07:15:02 CET 2012
[oli@bull]$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 44
model name      : Intel(R) Xeon(R) CPU           X5660  @ 2.80GHz
stepping        : 2
cpu MHz         : 2793.092
cache size      : 12288 KB
physical id     : 1
siblings        : 12
core id         : 0
cpu cores       : 6
apicid          : 32
fpu     : yes
fpu_exception   : yes
cpuid level     : 11
wp      : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm syscall nx pdpe1gb rdtscp lm constant_tsc ida nonstop_tsc arat pni monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr sse4_1 sse4_2 popcnt lahf_lm
bogomips        : 5586.18
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management: [8]


processor       : 1
...
<anologous data for processor 1--23>
## ------------------------------------------------------------

[oli@bull]$ cat /proc/meminfo
```

```
MemTotal:       24678244 kB
MemFree:        23567456 kB
Buffers:          405576 kB
Cached:           221364 kB
SwapCached:         1024 kB
Active:           281256 kB
Inactive:         493556 kB
HighTotal:             0 kB
HighFree:              0 kB
LowTotal:       24678244 kB
LowFree:        23567456 kB
SwapTotal:         16376 kB
SwapFree:           6912 kB
Dirty:               236 kB
Writeback:             0 kB
AnonPages:        146920 kB
Mapped:            13552 kB
Slab:             261624 kB
PageTables:         6152 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
CommitLimit:    12355496 kB
Committed_AS:    1048592 kB
VmallocTotal: 34359738367 kB
VmallocUsed:      263988 kB
VmallocChunk: 34359472995 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
Hugepagesize:       2048 kB
## ------------------------------------------------------------------------------------------
```