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

1 Preface 6

1.1 Changes with Respect to Previous Releases 6

2 The Benchmark Example: Client-Server Mutex 7

2.1 Criteria for Selecting the Example 7
2.2 Client-Server Mutex Protocol 7

3 Generation of Input Data 9

3.1 Generation of GTL formulation 9
3.2 Generation of SPIN/PROMELA input data 9
3.3 Generation of UPPAAL input data 9

4 The Measurement Process 11

4.1 Selection of Compile-Time / Run-Time Options 11
4.2 Graphical Presentation of Results 12

4.2.1 SPIN measurement with Compile=(none) Run=(none) 13
4.2.2 SPIN measurement with Compile=(none) Run=-a 14
4.2.3 SPIN measurement with Compile=(none) Run=-m999k 15
4.2.4 SPIN measurement with Compile=(none) Run=-m9999999k -a 16
4.2.5 SPIN measurement with Compile=-DSAFETY Run=-m9999999k 17
4.2.6 SPIN measurement with Compile=-O2 Run=-m9999999k -a 18
4.2.7 SPIN measurement with Compile=-O2 -DSAFETY Run=-m9999999k 19
4.2.8 SPIN measurement with Compile=-O3 Run=-m9999999k -a 20
4.2.9 SPIN measurement with Compile=-O3 -DSAFETY Run=-m9999999k 21
4.2.10 UPPAAL measurement with options -S 0 22
4.2.11 UPPAAL measurement with options -S 1 23
4.2.12 UPPAAL measurement with options -S 2 24
4.2.13 UPPAAL measurement with options \(-S\ 2\ -A\) ...................... 25
4.2.14 UPPAAL measurement with options \(-S\ 0\ -C\) ...................... 26
4.2.15 UPPAAL measurement with options \(-S\ 1\ -C\) ...................... 27
4.2.16 UPPAAL measurement with options \(-S\ 2\ -C\) ...................... 28
4.2.17 UPPAAL measurement with options \(-S\ 2\ -Z\) ...................... 29
4.2.18 UPPAAL measurement with options \(-S\ 2\ -n\ 0\) .................... 30
4.2.19 UPPAAL measurement with options \(-S\ 2\ -n\ 1\) .................... 31
4.2.20 UPPAAL measurement with options \(-S\ 2\ -n\ 2\) .................... 32
4.2.21 UPPAAL measurement with options \(-S\ 2\ -n\ 3\) .................... 33
4.2.22 UPPAAL measurement with options \(-S\ 2\ -n\ 4\) .................... 34
4.3 Tabular Presentation of Results ............................................. 35
  4.3.1 Time Consumption [s] .................................................. 36
  4.3.2 Memory Consumption [MB] .............................................. 37

5 Evaluation and Conclusion .................................................... 38

A Scripts and Utilities ............................................................. 41
  A.1 Script: gen-mutex-gtl.awk .............................................. 41
  A.2 Script: measure.bash ...................................................... 44
  A.3 Utility: plot.Makefile ..................................................... 48

B Generated Code ................................................................. 51
  B.1 Script-generated GTL models ............................................. 51
    B.1.1 GTL code for Mutex with 2 Clients ............................... 51
    B.1.2 GTL code for Mutex with 3 Clients ............................... 52
  B.2 SPIN/PROMELA Files Generated by the GTL Tool .................... 53
    B.2.1 PROMELA Model: Mutex with 2 Clients ............................ 53
    B.2.2 PROMELA Model: Mutex with 3 Clients ............................ 66
  B.3 UPPAAL XML Files Generated by the GTL-Tool ....................... 90
    B.3.1 UPPAAL Model: Mutex with 2 Clients ............................. 90
    B.3.2 UPPAAL Model: Mutex with 3 Clients ............................. 94
C Tool Information 100

C.1 GTL tool .................................................. 100
C.2 SPIN tool .................................................. 100
   C.2.1 C Compiler used to Compile SPIN Verifiers .............. 102
C.3 UPPAAL tool ............................................. 102

D Platform Information 105
List of Figures

4.1 Time Consumption [s] of SPIN with Various Compile-time and Run-time Options. .............................................. 36
4.2 Time Consumption [s] of UPPAAL with Various Run-time Options. .................................................. 36
4.3 Memory Allocation [MB] of SPIN with Various Compile-time and Run-time Options. .............................................. 37
4.4 Memory Allocation [MB] of UPPAAL with Various Run-time Options. .................................................. 37
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 project² — 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

<table>
<thead>
<tr>
<th>Issue</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>Supplemented BMBF Reference on title page.</td>
</tr>
<tr>
<td>1.1</td>
<td>Considered compile-time optimization options -02, -03 for SPIN. Removed some redundant UPPAAL plots (using -c or -z).</td>
</tr>
<tr>
<td>1.0</td>
<td>(Initial Release)</td>
</tr>
</tbody>
</table>

¹GTL: GALS translation language, where GALS stands for globally asynchronous, locally synchronous.
²Verifikation 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 $\text{mutex}_<N>.\text{gtl}$

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

This yields a $\text{mutex}_<N>\text{.pr}$ 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 $\text{mutex}_<N>.\text{gtl}$:

```
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 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).

¹Comparison 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.
²This explains why some numbers exceed the available 24GB of main memory without using swap.
For UPAL, there are no compile-time options available (to the user); it should be noted, however, that a 64-bit executable of UPAL is used, since this allows addressing of more than 4GB of memory.

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

<table>
<thead>
<tr>
<th>Compile Time</th>
<th>Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--a</code></td>
<td></td>
</tr>
<tr>
<td><code>--m999k</code></td>
<td></td>
</tr>
<tr>
<td><code>--m9999999k --a</code></td>
<td></td>
</tr>
<tr>
<td><code>-DSAFETY</code></td>
<td><code>--m9999999k</code></td>
</tr>
<tr>
<td><code>-O2</code></td>
<td><code>--m9999999k --a</code></td>
</tr>
<tr>
<td><code>-O2 -DSAFETY</code></td>
<td><code>--m9999999k</code></td>
</tr>
<tr>
<td><code>-O3</code></td>
<td><code>--m9999999k --a</code></td>
</tr>
<tr>
<td><code>-O3 -DSAFETY</code></td>
<td><code>--m9999999k</code></td>
</tr>
</tbody>
</table>

**SPIN option combinations used:**

<table>
<thead>
<tr>
<th>Compile Time</th>
<th>Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N/A)</td>
<td><code>--S 0</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 1</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 2</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 2 --A</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 0 --C</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 1 --C</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 2 --C</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 2 --Z</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 2 --n 0</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 2 --n 1</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 2 --n 2</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 2 --n 3</code></td>
</tr>
<tr>
<td>(N/A)</td>
<td><code>--S 2 --n 4</code></td>
</tr>
</tbody>
</table>

**UPAL option combinations used:**

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)

![Graph](image)

**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**

\[\text{Diagram showing user time in seconds and allocated main memory in GB} \]

**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=\(-m999k\)**

![Graph showing user time and allocated main memory measured with options: Compile= 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**

![Graph showing user time](image)

![Graph showing allocated main memory](image)
4.2.5 **SPIN measurement with**

**Compile**=--DSAFETY

**Run**=--m9999999k

---

**Graph 1:**

- **X-axis:** measured with options: Compile= -DSAFETY Run= -m9999999k
- **Y-axis:** user time [s] (logarithmic scale)

**Graph 2:**

- **X-axis:** measured with options: Compile= -DSAFETY Run= -m9999999k
- **Y-axis:** allocated main memory [GB] (logarithmic scale)
4.2.6 **SPIN measurement with Compile=\(-O2\) Run=\(-m9999999k\ -a\)**

![Graph showing user time and allocated main memory](image-url)
4.2.7 **SPIN measurement with Compile=\(-O2 -DSAFETY\) Run=\(-m9999999k\)**

![Graph showing user time and allocated main memory with logarithmic scale.](image)
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** \(-S\ 0\)

![Graph showing user time and allocated main memory with logarithmic scale](image)

- Measured with options `-S 0`.
- Uppaal64-4.1.7.
4.2.11 **UPPAAL measurement with options** \(-S 1\)
4.2.12 **UPPAAL measurement with options** $-S\ 2$

![Graph showing user time and allocated main memory with logarithmic scale](image-url)
4.2.13 **UPPAAL measurement with options** $-S \ 2 \ -A$

![Graph of user time (s) measured with options: -S 2 -A](image)

![Graph of allocated main memory (GB) measured with options: -S 2 -A](image)
4.2.14 **UPPAAL measurement with options** \(-S\ 0\ -C\)

![Graph showing user time and allocated main memory](image-url)
4.2.15 **UPPAAL measurement with options** \(-S\ 1\ -C\)

![Graph 1: User time (s) vs. system identifier](image1)

![Graph 2: Allocated main memory (GB) vs. system identifier](image2)
4.2.16 **UPPAAL measurement with options** `-S 2 -C`

![Graph of user time vs. options](image1)

![Graph of allocated main memory vs. options](image2)
4.2.17 **UPPAAL measurement with options** $-S \ 2 \ -Z$

![Graph showing user time (s) and allocated main memory (GB) with logarithmic scale]

**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`

![Graph 1: User time (s) versus measured options: -S 2 -n 0](image1)

![Graph 2: Allocated main memory (GB) versus measured options: -S 2 -n 0](image2)
4.2.19 **UPPAAL measurement with options** `-S 2 -n 1`

![Graph showing user time and allocated main memory measurements for UPPAAL with options `-S 2 -n 1`.

- **User Time [s] (logarithmic scale)**
  - Measured with options `-S 2 -n 1`.
  - UPPAAL 64-4.1.7.

- **Allocated Main Memory [GB] (logarithmic scale)**
  - Measured with options `-S 2 -n 1`.
  - UPPAAL 64-4.1.7.
4.2.20 UPPAAL measurement with options \(-S\ 2\ -n\ 2\)
4.2.21 **UPPAAL measurement with options** \(-S 2 -n 3\)

![Graph 1: User time [s] (logarithmic scale)](image1)

![Graph 2: Allocated main memory [GB] (logarithmic scale)](image2)
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]

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

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

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

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

<table>
<thead>
<tr>
<th>Compile</th>
<th>Run</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>20.90</td>
<td>22.25</td>
<td>32.18</td>
<td>100.62</td>
<td>566.45</td>
<td>2482.18</td>
<td>15477.00</td>
<td>(*)</td>
<td>20.90</td>
<td>22.25</td>
<td>32.18</td>
</tr>
<tr>
<td></td>
<td>-a</td>
<td>19.01</td>
<td>20.32</td>
<td>27.90</td>
<td>71.93</td>
<td>321.59</td>
<td>1261.00</td>
<td>16182.60</td>
<td>64105.17</td>
<td>(*)</td>
<td>-a</td>
<td>-a</td>
</tr>
<tr>
<td></td>
<td>-m999k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-m999999k -a</td>
<td>2155.03</td>
<td>2156.37</td>
<td>2166.32</td>
<td>2234.75</td>
<td>2718.60</td>
<td>6189.29</td>
<td>27875.21</td>
<td>(*)</td>
<td>-a</td>
<td>-a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-DSAFEY</td>
<td>-m999999k</td>
<td>2155.01</td>
<td>2156.34</td>
<td>2165.48</td>
<td>2234.65</td>
<td>2718.57</td>
<td>6188.85</td>
<td>27875.20</td>
<td>(*)</td>
<td>-a</td>
<td>-a</td>
</tr>
<tr>
<td></td>
<td>-O2</td>
<td>-m999999k</td>
<td>2154.93</td>
<td>2156.25</td>
<td>2166.17</td>
<td>2234.57</td>
<td>2718.42</td>
<td>6189.06</td>
<td>27874.93</td>
<td>(*)</td>
<td>-a</td>
<td>-a</td>
</tr>
<tr>
<td></td>
<td>-O2 -DSAFEY</td>
<td>-m999999k</td>
<td>2154.92</td>
<td>2156.23</td>
<td>2165.37</td>
<td>2234.50</td>
<td>2718.39</td>
<td>6188.62</td>
<td>27874.92</td>
<td>(*)</td>
<td>-a</td>
<td>-a</td>
</tr>
<tr>
<td></td>
<td>-O3</td>
<td>-m999999k</td>
<td>2155.01</td>
<td>2156.34</td>
<td>2165.28</td>
<td>2234.70</td>
<td>2718.56</td>
<td>6188.20</td>
<td>27875.10</td>
<td>(*)</td>
<td>-a</td>
<td>-a</td>
</tr>
<tr>
<td></td>
<td>-O3 -DSAFEY</td>
<td>-m999999k</td>
<td>2155.00</td>
<td>2156.32</td>
<td>2165.48</td>
<td>2234.62</td>
<td>2718.56</td>
<td>6188.78</td>
<td>27875.09</td>
<td>(*)</td>
<td>-a</td>
<td>-a</td>
</tr>
</tbody>
</table>

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

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

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 \( \approx 24 \) times faster when comparing best-to-best, without use of compiler optimizations (\(-02, -03\)) for SPIN the factor would be \( \approx 53 \).\(^1\)

   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 \( \approx 2.4 \) times faster when compiled with optimization. It is mildly surprising that \(-02\) actually performs a bit better than \(-03\). 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

\[
\frac{\sum_{N=2.8}^{8}(\text{time of successful SPIN runs})/(\text{number of successful SPIN runs})}{\sum_{N=2.8}^{8}(\text{time of successful UPPAAL runs})/(\text{number of successful UPPAAL runs})}
\]

\(^2\)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.\footnote{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.} 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.
Bibliography


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...
#

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\}_{\_acq};" >> filename
    print "_transitions\{\_acq;\}_{\_nc};" >> filename
    print "\}" >> filename
    print "\};" >> filename
    print "\}" >> filename
    print "\}

function fprint_false_array(filename, N, the_true_one){
    printf "[\" >> filename
```
# The code snippet provided is a part of a larger algorithm. It appears to be a
# program that includes a function to print a model server and some conditional
# checks. The code is written in C, with comments indicating the conditions and
# states of a system.

```c
for (i = 0; i < N; i++) {
    if (i > 0) { printf "," >> filename; }
    if (i == the_true_one) {
        printf "true" >> filename;
    } else {
        printf "false" >> filename;
    }
}
printf "]" >> filename;
}

function fprintf_server(filename, N) {
    printf "model\[\{" >> filename;
    printf "input enum\{nc,acq/cs,rel\}^" N "procstates;" >>filename;
    printf "output bool^" N "procouts;" >>filename;
    printf "\n" >>filename;
    printf "always false" >> filename;
    for (grant = 0; grant < N; grant++){
        for (j = 0; j < grant; j++){
            printf "procstates[" j "]!=\'acq\'and\" >> filename;
        }
        printf "procstates[ grant ]!=\'acq\'and\" >> filename;
        for (j = 0; j < N; j++){
            if (j != grant ){
                printf "procstates[" j "]!=\'cs\'and\" >> filename;
            }
        }
        printf "procouts=\" >> filename;
        fprintf_false_array(filename, N, grant);
        printf ")\n" >> filename;
    # (procstates[0] = \'acq\'and\procstates[1]!\'cs and procstates[2] != \'cs\'and
    # procouts = [true,false,false])
    # or (procstates[0] != \'acq\'and\procstates[1]!\'acq and procstates[0] !=
    # \!\'cs and procstates[2]!\'cs and procouts = [false,true,false])
    # or (procstates[0] != \'acq\'and\procstates[1]!\'acq and procstates[2] !=
    # \!\'acq and procstates[0]!\'acq and procstates[1] != \'cs and procouts!=[
    # \!false ,false ,true])
    printf \"or\(procouts=\" >> filename;
    fprintf_false_array(filename, N, -1);
    printf ");\n" >> filename;
}
print "}\" >>filename;
}
BEGIN {
    N = ARGV[1] + 0;
    if (N < 1) {
        print "usage: gen-mutext-gtl.awk<N>\n"
    }
```

This code is part of a larger system, likely for verification purposes, involving conditional logic to decide the state of a system based on certain conditions.
exit 1;
}

print "##_generating_for:_N=_" N;

filename = "./mutex_" N ".gtl";
print "##_output_gtl_file=_" filename;
print "/GENERATED_BY_gen-mutex-gtl.awk" > filename

fprint_client(filename);
fprint_server(filename, N);

print 
"\n//_instances_" >> filename
for ( i = 0; i < N; i++) {
print "instance_client_c" i ";" >> filename
}
print "instance_server_s;" >> filename

print 
"\n//_connections_" >> filename
for ( i = 0; i < N; i++) {
print "connect_c" i ".st_s.procsstates[i]" >> filename
}
for ( i = 0; i < N; i++) {
print "connect_s.procouts[i].c" i ".proceed;" >> filename
}

print 
"\n//_safety_claim_" >> filename;
print "verify_" >> filename;
for ( i = 0; i < N; i++){
printf "always(c)" i ".st_s.cs=!false" >> filename;
for ( j = 0; j < N; j++){
if ( i != j){
    printf "or_c" j ".st_s.cs_c" >> filename;
}
}
printf "\n" >> filename;
}

query_file = "/mutex_" N ".q";

print "##_output_query_file=" query_file;
print "/GENERATED_BY_gen-mutex-gtl.awk" > query_file;
printf "A[(not(.false_c)) _](c) j ".11_&c" i ".11_" >> query_file;
for ( i = 0; i < N; i++) {
    printf "\n" >> query_file;
}
exit 0;
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
```

```sh
#!/bin/bash
#
# $#Id: measure.bash 1843 2012-02-06 09:18:34Z moeller $
# $# $URL: svn://theo.it.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\nuser\nsys\nMemory\nSwaps" $@

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

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

```
FILE: brief_info_time.txt
```

```txt
[oli@bull measured.bull]$ /usr/bin/time --version
GNU time 1.7
man time
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 statistics consist of (i) the elapsed real time between invocation and termination, (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
CPU time (the sum of the tms_stime and tms_cstime values in a struct
tms as returned by times(2)).

Note: some shells (e.g., bash(1)) have a built-in time command that
provides less functionality than the command described here. To access
the real command, you may need to specify its pathname (something like
/usr/bin/time).

OPTION
−p When in the POSIX locale, use the precise traditional format
"real %f\nuser %f\nsys %f\n"
(with numbers in seconds) where the number of decimals in the
output for %f is unspecified but is sufficient to express the
clock tick accuracy, and at least one.

ENVIRONMENT
The variables LANG, LC_ALL, LC_CTYPE, LC_MESSAGES, LC_NUMERIC, NLSPATH
and PATH are used. The last one to search for command. The remaining
ones for the text and formatting of the output.

EXIT STATUS
If command was invoked, the exit status is that of command. Otherwise
it is 127 if command could not be found, 126 if it could be found but
could not be invoked, and some other non-zero value (1−125) if some-
thing else went wrong.

SEE ALSO
times(2)

GNU VERSION
Below a description of the GNU 1.7 version of time. Disregarding the
name of the utility, GNU makes it output lots of useful information,
not only about time used, but also on other resources like memory, I/O
and IPC calls (where available). The output is formatted using a for-
mat string that can be specified using the −f option or the TIME envi-
ronment variable.

The default format string is
%Uuser %Ssystem %Eelapsed %PCPU (%Xtext+%Ddata %Mmax)k
%I inputs+%Ooutputs (%Fmajor+%Rminor) pagefaults %Wswaps

When the −p option is given the (portable) output format
  real %e
  user %U
  sys %S
is used.

The format string
The format is interpreted in the usual printf-like way. Ordinary char-
acters are directly copied, tab, newline and backslash are escaped
using \t, \n and \\
, a percent sign is represented by %%, and otherwise
% indicates a conversion. The program time will always add a trailing
newline itself. The conversions follow. All of those used by tcsh(1)
are supported.

Time

\%E\hspace{1em}Elapsed real time (in hours:minutes:seconds).

\%e\hspace{1em}(Not in tcsh.) Elapsed real time (in seconds).

\%S\hspace{1em}Total number of CPU–seconds that the process spent in kernel mode.

\%U\hspace{1em}Total number of CPU–seconds that the process spent in user mode.

\%P\hspace{1em}Percentage of the CPU that this job got, computed as (\%U + \%S) / \%E.

Memory

\%M\hspace{1em}Maximum resident set size of the process during its lifetime, in Kbytes.

\%t\hspace{1em}(Not in tcsh.) Average resident set size of the process, in Kbytes.

\%K\hspace{1em}Average total (data+stack+text) memory use of the process, in Kbytes.

\%D\hspace{1em}Average size of the process’s unshared data area, in Kbytes.

\%p\hspace{1em}(Not in tcsh.) Average size of the process’s unshared stack space, in Kbytes.

\%X\hspace{1em}Average size of the process’s shared text space, in Kbytes.

\%Z\hspace{1em}(Not in tcsh.) System’s page size, in bytes. This is a per–system constant, but varies between systems.

\%F\hspace{1em}Number of major page faults that occurred while the process was running. These are faults where the page has to be read in from disk.

\%R\hspace{1em}Number of minor, or recoverable, page faults. These are faults for pages that are not valid but which have not yet been claimed by other virtual pages. Thus the data in the page is still valid but the system tables must be updated.

\%W\hspace{1em}Number of times the process was swapped out of main memory.

\%c\hspace{1em}Number of times the process was context–switched involuntarily (because the time slice expired).

\%w\hspace{1em}Number of waits: times that the program was context–switched voluntarily, for instance while waiting for an I/O operation to complete.
I/O

%I Number of file system inputs by the process.

%O Number of file system outputs by the process.

%r Number of socket messages received by the process.

%s Number of socket messages sent by the process.

%k Number of signals delivered to the process.

%C (Not in tcsh.) Name and command line arguments of the command being timed.

%x (Not in tcsh.) Exit status of the command.

GNU OPTIONS

-f FORMAT, --format=FORMAT
   Specify output format, possibly overriding the format specified
   in the environment variable TIME.

-p, --portability
   Use the portable output format.

-o FILE, --output=FILE
   Do not send the results to stderr, but overwrite the specified file.

-a, --append
   (Used together with -o.) Do not overwrite but append.

-v, --verbose
   Give very verbose output about all the program knows about.

GNU STANDARD OPTIONS

--help Print a usage message on standard output and exit successfully.

-V, --version
   Print version information on standard output, then exit successfully.

-- Terminate option list.

BUGS

Not all resources are measured by all versions of Unix, so some of the
values might be reported as zero. The present selection was mostly
inspired by the data provided by 4.2 or 4.3BSD.

GNU time version 1.7 is not yet localized. Thus, it does not implement
the POSIX requirements.

The environment variable TIME was badly chosen. It is not unusual for
systems like autoconf or make to use environment variables with the
name of a utility to override the utility to be used. Uses like MORE or
TIME for options to programs (instead of program pathnames) tend to lead to difficulties.

It seems unfortunate that `-o` overwrites instead of appends. (That is, the `-a` option should be the default.)

Mail suggestions and bug reports for GNU time to

bug-utils@prep.ai.mit.edu

Please include the version of time, which you can get by running

time --version

and the operating system and C compiler you used.

SEE ALSO
tcsh(1), times(2), wait3(2)

AUTHOR

David Keppel

Original version

David MacKenzie

POSIXization, autoconfiscation, GNU getoptimization, documentation, other bug fixes and improvements.

Arne Henrik Juul

Helped with portability

Francois Pinard

Helped with portability

2000–12–11

A.3 Utility: plot.Makefile

This Makefile was used to generate the plots.

```
FILE: utils/plot.Makefile

###
### $RCSfile: plot.Makefile,v $ $Revision: 1.3 $
###
ROOT:=$( shell readlink -f 'pwd'/../..
BASENAME:=$( shell pwd | sed 's+.*/[SU]MEASURE.*+[g]')
BASEPDFNAME:=$( shell echo $(BASENAME) | sed 's+\.[^]+_[^]+g')
DATS:=time.dat memory.dat
TOOL_CHAR:=$( shell pwd | sed 's+.*\([SU]\)\MEASURE\.[^]+\+\+g')
```
ifeq ($(TOOL_CHAR),U)
  TOOL:=Uppaal
  TITLE:=Uppaal64 – 4.1.7
  COLOR:=blue
  OPTIONS:=$(shell echo $(BASENAME) | sed 's/\_/\_/g')
else
  TOOL:=Spin
  TITLE:=Spin – 6.1.0
  COLOR:=red
  OPTIONS:=(shell echo $(BASENAME) | cut –d’.’ –f 1| sed 's/\_/\_/g')
endif

clean:
  rm -f *.gpl
  rm -f *.ps
  rm -f *.pdf
  rm -f *.dat

%.gpl: %$(BASENAME).gpl $(ROOT)/%COMMON.gpl
  cat $(<) \n    | sed ’s/@BASENAME@/$(BASENAME)/g’ \n    | sed ’s/@COLOR@/$(COLOR)/g’ \n    | sed ’s/@TITLE@/$(TITLE)/g’ \n    | sed ’s/@OPTIONS@/$(OPTIONS)/g’ \n  > $(@)

%.ps: %.gpl
  gnuplot $(<)

%.pdf: %.ps
  ps2pdf $(<) $(@)

%.bb: %.ps
  grep BoundingBox $(<) >$(@)

time.dat: $(wildcard *.log)
  $(ROOT)/utils/get_time.bash >$(@)

time_real.dat: $(wildcard *.log)
  $(ROOT)/utils/get_time.bash real >$(@)

memory.dat: $(wildcard *.log)
  $(ROOT)/utils/get_memory_MB.bash >$(@)

#
#

all: $(DATS) $(PLOTS)

install: $(PLOTS)
  cp $(PLOTS) $(ROOT)/PLOTS/

##
.PHONEY: debug show

  show: all
    xpdf -z page plot_time_(BASENAME).pdf &
    xpdf -z page plot_memory_(BASENAME).pdf &

debug:
    @echo ROOT=$(ROOT)
    @echo BASENAME=$(BASENAME)
    @echo BASEPDFNAME=$(BASEPDFNAME)
    @echo TOOL_CHAR=$(TOOL_CHAR)
    @echo TOOL=$(TOOL)
    @echo TITLE=$(TITLE)
    @echo COLOR=$(COLOR)
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

```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]);
```
B.1.2 GTL code for Mutex with 3 Clients

**FILE: mutex_3.gtl**

```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;
  // connections
  connect c0 . s . procstates[0];
  connect c1 . 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'));
  }
}
```
init procouts [false, false, false];
always false
       and procouts = [true, false, false])
   or (procstates[0] != 'acq and procstates[1] = 'acq and procstates[0] != 'cs
       and procstates[2] != 'cs and procouts = [false, true, false])
       and procstates[0] != 'cs and procstates[1] != 'cs and procouts = [false, false, true])
or (procouts = [false, false, false]);
}

// instances
instance client c0;
instance client c1;
instance client c2;
instance server s;

// connections
connect c0.st s.procstates[0];
connect c1.st s.procstates[1];
connect c2.st s.procstates[2];
connect s.procouts[0] c0.proceed;
connect s.procouts[1] c1.proceed;
connect s.procouts[2] c2.proceed;

// safety claim
verify {
   always (c0.st = 'cs => !(false or c1.st = 'cs or c2.st = 'cs));
   always (c1.st = 'cs => !(false or c0.st = 'cs or c2.st = 'cs));
   always (c2.st = 'cs => !(false or c0.st = 'cs or c1.st = 'cs));
}

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>.glt

B.2.1 PROMELA Model: Mutex with 2 Clients

FILE: mutex_2.pr

bool c0_proceed_0 = false
int c0_st_0 = 0
bool c1_proceed_0 = false
int c1_st_0 = 0
int s_procstates_0_0 = 0
int s_procstates_1_0 = 0
int _count_c0, _count_c1, _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\n\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
    }
}

if :: atomic {
    _count_c0 == 0;
    if :: s_procstates_0_0 = 0;
        c0_st_0 = s_procstates_0_0
    fi;
    printf("TRANSITION_c0\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;
st0:
    if :: atomic {
        _count_c0 == 0 && c0_proceed_0;
        if :: s_procstates_0_0 = 2;
            c0_st_0 = s_procstates_0_0
        fi;
        printf("TRANSITION\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
                _count_c0 = _count_c0 - _minimum;
                _count_c1 = _count_c1 - _minimum;
                _count_s = _count_s - _minimum
            fi;
            goto st1
        }
    }
    :: atomic {
        _count_c0 == 0 && ! (c0_proceed_0);
        if :: s_procstates_0_0 = 1;
            c0_st_0 = s_procstates_0_0
        fi;
        printf("TRANSITION\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
                _count_c0 = _count_c0 - _minimum;
                _count_c1 = _count_c1 - _minimum;
                _count_s = _count_s - _minimum
            fi;
            goto st0
        }
    }
    fi;
st1:
  if
    :: atomic {
      _count_c0 == 0;
      if
        :: s_procstates_0_0 = 3;
        c0_st_0 = s_procstates_0_0
      fi;
      printf("TRANSITION_c0_1_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 st3
    };
  fi;

:: atomic {
  _count_c0 == 0;
  if
    :: s_procstates_0_0 = 2;
    c0_st_0 = s_procstates_0_0
  fi;
  printf("TRANSITION_c0_1_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 st1
}
st2:
  :: 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;
st3:
    if
        :: atomic {
            _count_c0 == 0;
            if
                :: s_procstates_0_0 = 0;
                c0_st_0 = s_procstates_0_0
            fi;
            printf("TRANSITION_c0_3_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 st2
        }
    fi
}
proctype c1() {
    if
        :: atomic {
            _count_c1 == 0;
            if
                :: s_procstates_1_0 = 1;
                c1_st_0 = s_procstates_1_0
            fi;
            printf("TRANSITION_c1_2_0\n");
            d_step {
                _count_c1 = _count_c1 + 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_c1 == 0;
    if
        :: s_procstates_1_0 = 0;
        c1_st_0 = s_procstates_1_0
    fi;
    printf("TRANSITION\_c1\_2\_1\n");
    d_step {
        _count_c1 = _count_c1 + 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;

st0:
    if
        :: atomic {
            _count_c1 == 0 &\& c1_proceed_0;
            if
                :: s_procstates_1_0 = 2;
                c1_st_0 = s_procstates_1_0
            fi;
            printf("TRANSITION\_c1\_0\_2\_0\n");
            d_step {
                _count_c1 = _count_c1 + 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
        }:: atomic {
    fi;

Verified Systems International GmbH
};
  goto st1
}
:: atomic {
  _count_c1 == 0 && !(c1_proceed_0);
  if
  :: s_procstates_1_0 = 1;
  c1_st_0 = s_procstates_1_0
  fi;
  printf("TRANSITION \text{c1} \_0 \rightarrow \_1\n");
  d_step {
    _count_c1 = _count_c1 + 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
}

fi;
st1:
  :: atomic {
    _count_c1 == 0;
    if
    :: s_procstates_1_0 = 3;
    c1_st_0 = s_procstates_1_0
    fi;
    printf("TRANSITION \text{c1} \_1 \rightarrow \_0\n");
    d_step {
      _count_c1 = _count_c1 + 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 st3
}
:: atomic {
  _count_c1 == 0;
  if
    :: s_procstates_1_0 = 2;
    c1_st_0 = s_procstates_1_0
  fi;
  printf("TRANSITION\n\n");
  d_step {
    _count_c1 = _count_c1 + 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 st1
  fi;
}
st2:
  if
    :: atomic {
    _count_c1 == 0;
    if
      :: s_procstates_1_0 = 1;
      c1_st_0 = s_procstates_1_0
    fi;
    printf("TRANSITION\n\n");
    d_step {
      _count_c1 = _count_c1 + 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
  

Verified Systems International GmbH

61
};

goto st0

:: atomic {
  _count_c1 == 0;
  if
    :: s_procstates_1_0 = 0;
    c1_st_0 = s_procstates_1_0
  fi;
  printf("TRANSITION_c1_2_1\n");
  d_step {
    _count_c1 = _count_c1 + 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;

st3:
if
  :: atomic {
    _count_c1 == 0;
    if
      :: s_procstates_1_0 = 0;
      c1_st_0 = s_procstates_1_0
    fi;
    printf("TRANSITION_c1_3_0\n");
    d_step {
      _count_c1 = _count_c1 + 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
      }
proctype s() {
    if :: atomic {
        _count_s == 0 && s_procstates_0_0 != 1 && s_procstates_1_0 == 1 && 
        s_procstates_0_0 != 2;
        c0_proceed_0 = false;
        c1_proceed_0 = true;
        printf("TRANSITION\0\0\0\n");
        d_step {
            _count_s = _count_s + 1;
            _minimum = _count_c0;
            if :: _count_c1 < _minimum;
                _minimum = _count_c1
            :: else
                fi; 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_s == 0;
        c0_proceed_0 = false;
        c1_proceed_0 = false;
        printf("TRANSITION\0\0\0\n");
        d_step {
            _count_s = _count_s + 1;
            _minimum = _count_c0;
            if :: _count_c1 < _minimum;
                _minimum = _count_c1
            :: else
                fi; 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
    }
};
goto st2
:: atomic {
    _count_s = 0 && s_procstates_0_0 == 1 && s_procstates_1_0 != 2;
    c0_proceed_0 = true;
    c1_proceed_0 = false;
    printf("TRANSITION\n"");
    d_step {
        _count_s = _count_s + 1;
        _minimum = _count_c0;
        if
            :: _count_c1 < _minimum;
        :: else
            _minimum = _count_c1
        fi;
        if
            :: _count_s < _minimum;
        :: else
            _minimum = _count_s
        fi;
        _count_c0 = _count_c0 - _minimum;
        _count_c1 = _count_c1 - _minimum;
        _count_s = _count_s - _minimum
    };
    goto st0
} fi;

st0:
    if
        :: atomic {
            _count_s = 0 && s_procstates_0_0 != 1 && s_procstates_1_0 == 1 &&
            ~s_procstates_0_0 != 2;
            c0_proceed_0 = false;
            c1_proceed_0 = true;
            printf("TRANSITION\n"");
            d_step {
                _count_s = _count_s + 1;
                _minimum = _count_c0;
                if
                    :: _count_c1 < _minimum;
                :: else
                    _minimum = _count_c1
                fi;
                if
                    :: _count_s < _minimum;
                :: else
                    _minimum = _count_s
                fi;
                _count_c0 = _count_c0 - _minimum;
                _count_c1 = _count_c1 - _minimum;
                _count_s = _count_s - _minimum
            };
            goto st0
        } else
            :: atomic {
                _count_s = 0;
            }
c0_proceed_0 = false;
c1_proceed_0 = false;
printf("TRANSITION_s_0_1\n");
d_step {
    _count_s = _count_s + 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_s == 0 && s_procstates_0_0 == 1 && s_procstates_1_0 != 2;
c0_proceed_0 = true;
c1_proceed_0 = false;
printf("TRANSITION_s_0_2\n");
d_step {
    _count_s = _count_s + 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
}
```c
} never {
  if :: atomic {
    goto st0
  }
  :: atomic {
    c1_st_0 == 2 && c0_st_0 == 2;
    goto st2
  }
  :: atomic {
    goto st1
  }
  :: atomic {
    c0_st_0 == 2 && c1_st_0 == 2;
    goto st2
  }
  fi;

st0:
  if :: atomic {
    goto st0
  }
  :: atomic {
    c1_st_0 == 2 && c0_st_0 == 2;
    goto st2
  }
  fi;

st1:
  if :: atomic {
    goto st1
  }
  :: atomic {
    c0_st_0 == 2 && c1_st_0 == 2;
    goto st2
  }
  fi;

accept2:
  st2:
  if :: atomic {
    goto st2
  }
  fi
}
```

### B.2.2 PROMELA Model: Mutex with 3 Clients

**FILE: mutex_3.pr**

```plaintext
bool c0_proceed_0 = false
int c0_st_0 = 0
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() {
    :: 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
:: 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 st2
} fi;

st0:
if
  atomic {
    _count_c0 == 0 && c0_proceed_0;
    if
      :: s_procstates_0_0 = 2;
      c0_st_0 = s_procstates_0_0
    fi;
    printf("TRANSITION_c0_0_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 st1
  }
}
```c
_count_c0 == 0 && ! (c0_proceed_0);
if
  :: s_procstates_0_0 = 1;
  c0_st_0 = s_procstates_0_0
fi;
printf("TRANSITION_c0_0_0_1\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
}

fi;
st1:
if
  :: atomic {
    _count_c0 == 0;
    if
      :: s_procstates_0_0 = 3;
      c0_st_0 = s_procstates_0_0
    fi;
    printf("TRANSITION_c0_1_1\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;
    }
  }
```
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};
181
:: atomic {
182  _count_c0 == 0;
183  if
184    :: s_procstates_0_0 = 2;
185    c0_st_0 = s_procstates_0_0
186  fi;
187  printf("TRANSITION_c0_1_1\n");
188  d_step {
189    _count_c0 = _count_c0 + 1;
190    _minimum = _count_c0;
191    if
192      :: _count_c1 < _minimum;
193      _minimum = _count_c1
194    :: else
195    fi;
196    if
197      :: _count_c2 < _minimum;
198      _minimum = _count_c2
199    :: else
200    fi;
201    if
202      :: _count_s < _minimum;
203      _minimum = _count_s
204    :: else
205    fi;
206    _count_c0 = _count_c0 - _minimum;
207    _count_c1 = _count_c1 - _minimum;
208    _count_c2 = _count_c2 - _minimum;
209    _count_s = _count_s - _minimum
210};
211  goto st1
212}
213

:: atomic {
214  _count_c0 == 0;
215  if
216    :: s_procstates_0_0 = 1;
217    c0_st_0 = s_procstates_0_0
218  fi;
219  printf("TRANSITION_c0_2_0\n");
220  d_step {
221    _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
  :
    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 st2
}
st3:
  if :: atomic {
    _count_c0 == 0;
    if :: s_procstates_0_0 = 0;
      c0_st_0 = s_procstates_0_0
    fi;
    printf("TRANSITION_c0\n\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 st2
  }
}
proctype c1() {
  if :: atomic {
    _count_c1 == 0;
    if :: s_procstates_1_0 = 1;
      c1_st_0 = s_procstates_1_0
    fi;
    printf("TRANSITION_c1\n\n");
    d_step {
      _count_c1 = _count_c1 + 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
};
go to st0
:: atomic {
  _count_c1 == 0;
  if :
    :: s_proccstates_1_0 = 0;
    c1_st_0 = s_proccstates_1_0
  fi;
  printf("TRANSITION_c1_2_1\n");
d_step {
  _count_c1 = _count_c1 + 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 st2
  }
fi;
st0:
if :
  :: atomic {
    _count_c1 == 0 && c1_proceed_0;
    if :
      :: s_proccstates_1_0 = 2;
c1_st_0 = s_procstates_1_0
fi;
printf("TRANSITION_c1_0_0\n");
d_step {
  _count_c1 = _count_c1 + 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 st1

:: atomic {
  _count_c1 == 0 && !(c1_proceed_0);
  if
    :: s_procstates_1_0 = 1;
    c1_st_0 = s_procstates_1_0
  fi;
  printf("TRANSITION_c1_0_1\n");
d_step {
  _count_c1 = _count_c1 + 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;
```c
_count_c2 = _count_c2 - _minimum;
_count_s = _count_s - _minimum
};
goto st0
}
fi;
goto st0
}
st1:
if
    :: atomic {
        _count_c1 == 0;
        :: s_procstates_1_0 = 3;
        c1_st_0 = s_procstates_1_0
        fi;
        printf("TRANSITION\n");
    d_step {
        _count_c1 = _count_c1 + 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 st3
            }
        }
    }
```
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
};
go to st1
}
fi;
st2 :
if
   : atomic {
   _count_c1 == 0;
   if
      : s_procstates_1_0 = 1;
      c1_st_0 = s_procstates_1_0
   fi;
   printf("TRANSITION_c1_2_0\n");
d_step {
   _count_c1 = _count_c1 + 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
};
go to st0
}
: atomic {
   _count_c1 == 0;
   if

.readyState = s_procstates_1_0;
c1_st_0 = s_procstates_1_0
fi;
printf("TRANSITION\n\n");
d_step {
  _count_c1 = _count_c1 + 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 st2
} fi;
st3:
if
  :: atomic {
    _count_c1 == 0;
    if
      :: s_procstates_1_0 == 0;
      c1_st_0 = s_procstates_1_0
    fi;
    printf("TRANSITION\n\n");
    d_step {
      _count_c1 = _count_c1 + 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;
    };
  }
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 }
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
");
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
");
661     d_step {
662       _count_c2 = _count_c2 + 1;
663       _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 st2
}
fi;
st0:
if
  :: atomic {
    _count_c2 == 0 && c2_proceed_0;
    if
      :: s_procstates_2_0 = 2;
      c2_st_0 = s_procstates_2_0
    fi;
    printf("TRANSITION_c2_0_0\n");
    d_step {
      _count_c2 = _count_c2 + 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 st1

:: atomic {
    _count_c2 == 0 && !(c2_proceed_0);
    if
        :: s_procstates_2_0 = 1;
        c2_st_0 = s_procstates_2_0
    fi;
    printf("TRANSITION\text{c2}_0\_0\_1\n");
    d_step {
        _count_c2 = _count_c2 + 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
}

fi;

st1:

if
    :: atomic {
        _count_c2 == 0;
    if
        :: s_procstates_2_0 = 3;
        c2_st_0 = s_procstates_2_0
    fi;
    printf("TRANSITION\text{c2}_1\_0\_0\n");
    d_step {
        _count_c2 = _count_c2 + 1;
        _minimum = _count_c0;
        if
            :: _count_c1 < _minimum;
            _minimum = _count_c1
        :: else
            fi;
        if
            :: _count_c2 < _minimum;
            _minimum = _count_c2

Summary:
The code snippet represents a transition logic that uses a combination of conditional checks and arithmetic operations to determine the state transitions of a system. It involves variables such as `_count_c0`, `_count_c1`, `_count_c2`, and `_count_s` along with the `c2_proceed_0` variable. The transitions are controlled by conditional statements that update these variables according to specific rules.
```c
:: 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 st3
}
:: atomic {
    _count_c2 == 0;
if
    :: s_procstates_2_0 = 2;
c2_st_0 = s_procstates_2_0
fi;
printf("TRANSITION c2_1,1\n");
d_step {
    _count_c2 = _count_c2 + 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 st1
} fi;
st2:
if
    :: atomic {
    _count_c2 == 0;
if
    :: s_procstates_2_0 = 1;
c2_st_0 = s_procstates_2_0
fi;
```
printf("TRANSITION_c2_2_0\n");
d_step {
    _count_c2 = _count_c2 + 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_c2 == 0;
    if
        :: s_procstates_2_0 == 0;
        c2_st_0 = s_procstates_2_0
    fi;
    printf("TRANSITION_c2_2_1\n");
d_step {
    _count_c2 = _count_c2 + 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 st2
  }
fi;
st3:
if
  :: atomic {
    _count_c2 == 0;
    if
      :: s_procstates_2_0 = 0;
      c2_st_0 = s_procstates_2_0
    fi;
    printf("TRANSITION_c2_3\n");
d_step {
    _count_c2 = _count_c2 + 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 st2
  fi
}
proctype s() {
  if
    :: atomic {
      _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;
      c0_proceed_0 = false;
      c1_proceed_0 = false;
      c2_proceed_0 = true;
      printf("TRANSITION_s_0\n");
d_step {
    _count_s = _count_s + 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 st2
}}
:: 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_s == 0 && s_procstates_0_0 == 1 && s_procstates_1_0 != 2 && ~s_procstates_2_0 != 2;
c0_proceed_0 = true;
c1_proceed_0 = false;
c2_proceed_0 = false;
printf("TRANSITION\n1n");
d_step {
    _count_s = _count_s + 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_s == 0;
c0_proceed_0 = false;
c1_proceed_0 = false;
c2_proceed_0 = false;
printf("TRANSITION_s_0_2\n");
d_step {
    _count_s = _count_s + 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_s == 0 && s_procstates_0_0 != 1 && s_procstates_1_0 == 1 && __s_procstates_0_0 != 2 && s_procstates_2_0 != 2;
    c0_proceed_0 = false;
    c1_proceed_0 = true;
    c2_proceed_0 = false;
    printf("TRANSITION_s_0_3\n");
d_step {
    _count_s = _count_s + 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
fi;
st0:
if :: atomic {
  _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;
  c0_proceed_0 = false;
  c1_proceed_0 = false;
  c2_proceed_0 = true;
  printf("TRANSITION_s_0_0\n")
  d_step {
    _count_s = _count_s + 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
}
if :: atomic {
  _count_s == 0 && s_procstates_0_0 == 1 && s_procstates_1_0 != 2 && _
  _s_procstates_2_0 != 2;
  c0_proceed_0 = true;
  c1_proceed_0 = false;
  c2_proceed_0 = false;
  printf("TRANSITION_s_0_1\n")
  d_step {
    _count_s = _count_s + 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_s == 0;
c0_proceed_0 = false;
c1_proceed_0 = false;
c2_proceed_0 = false;
printf("TRANSITION_s\_0\_2\n");

d_step {
  _count_s = _count_s + 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_s == 0 && s_procstates_0_0 != 1 && s_procstates_1_0 == 1 && s_procstates_0_0 != 2 && s_procstates_2_0 == 1
  c0_proceed_0 = false;
c1_proceed_0 = true;
c2_proceed_0 = false;
printf("TRANSITION_s\_0\_3\n");
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  }
1178  init {  
1179    atomic {
1180      skip;
1181      run c0();
1182      run c1();
1183      run c2();
1184      run s()
1185    }
1186  }
1187  never {  
1188    if
1189    :: atomic {
1190      c2_st_0 == 2 && c0_st_0 == 2;  
1191      goto st3
1192    }  
1193    :: atomic {
1194      goto st0
1195    }
1196    :: atomic {
1197      c1_st_0 == 2 && c0_st_0 == 2;  
1198      goto st3
1199    }  
1200    :: atomic {
1201      c2_st_0 == 2 && c1_st_0 == 2;  
1202      goto st3
1203    }  
1204    :: atomic {
1205      goto st1
1206    }
:: atomic {
    c0_st_0 == 2 && c1_st_0 == 2;
    goto st3
}
:: atomic {
    goto st2
}
:: atomic {
    c0_st_0 == 2 && c2_st_0 == 2;
    goto st3
}
:: atomic {
    c1_st_0 == 2 && c2_st_0 == 2;
    goto st3
}
:: atomic {
    c0_st_0 == 2 && c2_st_0 == 2;
    goto st3
}
fi;

st0:
    if
    :: atomic {
        c2_st_0 == 2 && c0_st_0 == 2;
        goto st3
    }
    :: atomic {
        goto st0
    }
    :: atomic {
        c1_st_0 == 2 && c0_st_0 == 2;
        goto st3
    }
    fi;

st1:
    if
    :: atomic {
        c2_st_0 == 2 && c1_st_0 == 2;
        goto st3
    }
    :: atomic {
        goto st1
    }
    :: atomic {
        c0_st_0 == 2 && c1_st_0 == 2;
        goto st3
    }
    fi;

st2:
    if
    :: atomic {
        c1_st_0 == 2 && c2_st_0 == 2;
        goto st3
    }
    :: atomic {
        goto st2
    }
    :: atomic {
        c0_st_0 == 2 && c2_st_0 == 2;
    }
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>.gtl (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

<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[0,3] s_procsstates_0[1] = {0};
int[0,3] s_procsstates_1[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="10"/>
  <label kind="assignment">s_procsstates_0[0] = 1, c0_st[0] = 1</label>
</transition>

<transition>
  <source ref="start"/>
  <target ref="12"/>
  <label kind="assignment">s_procsstates_0[0] = 0, c0_st[0] = 0</label>
</transition>

<transition>
  <source ref="10"/>
  <target ref="11"/>
  <label kind="guard">c0_proceed[0]</label>
  <label kind="assignment">s_procsstates_0[0] = 2, c0_st[0] = 2</label>
</transition>

<transition>
  <source ref="11"/>
  <target ref="13"/>
  <label kind="assignment">s_procsstates_0[0] = 3, c0_st[0] = 3</label>
</transition>

<transition>
  <source ref="11"/>
  <target ref="11"/>
  <label kind="assignment">s_procsstates_0[0] = 2, c0_st[0] = 2</label>
</transition>

<transition>
  <source ref="12"/>
  <target ref="10"/>
  <label kind="assignment">s_procsstates_0[0] = 1, c0_st[0] = 1</label>
</transition>

<transition>
  <source ref="12"/>
  <target ref="12"/>
  <label kind="assignment">s_procsstates_0[0] = 0, c0_st[0] = 0</label>
</transition>

<transition>
  <source ref="13"/>
  <target ref="12"/>
  <label kind="assignment">s_procsstates_0[0] = 0, c0_st[0] = 0</label>
</transition>

</template>

<template>
  <name>c1 tmpl</name>
  <location id="start">
    <name>start</name>
    <urgent/>
  </location>
  <location id="10">
  </location>
</template>
<init ref="start"/>
<transition>
  <source ref="start"/>
  <target ref="l0"/>
  <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
</transition>
<transition>
  <source ref="start"/>
  <target ref="l2"/>
  <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
</transition>
<transition>
  <source ref="l0"/>
  <target ref="l1"/>
  <label kind="guard">c1_proceed[0]</label>
  <label kind="assignment">s_procstates_1[0] = 2, c1_st[0] = 2</label>
</transition>
<transition>
  <source ref="l0"/>
  <target ref="l0"/>
  <label kind="guard">!c1_proceed[0]</label>
  <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
</transition>
<transition>
  <source ref="l1"/>
  <target ref="l3"/>
  <label kind="assignment">s_procstates_1[0] = 3, c1_st[0] = 3</label>
</transition>
<transition>
  <source ref="l1"/>
  <target ref="l1"/>
  <label kind="assignment">s_procstates_1[0] = 2, c1_st[0] = 2</label>
</transition>
<transition>
  <source ref="l2"/>
  <target ref="l0"/>
  <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
</transition>
<transition>
  <source ref="l2"/>
  <target ref="l2"/>
  <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
</transition>
<source ref="13"/>
<target ref="12"/>
<label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
</transition>
</template>
<template>
  <name>s_tmpl</name>
  <location id="start">
    <name>start</name>
    <urgent/>
  </location>
  <location id="l0">
    <name>l0</name>
  </location>
  <init ref="start"/>
  <transition>
    <source ref="start"/>
    <target ref="l0"/>
    <label kind="guard">s_procstates_0[0] != 1 and s_procstates_1[0] == 1 and s_procstates_0[0] != 2</label>
    <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 1</label>
  </transition>
  <transition>
    <source ref="start"/>
    <target ref="l0"/>
    <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0</label>
  </transition>
  <transition>
    <source ref="l0"/>
    <target ref="l0"/>
    <label kind="guard">s_procstates_0[0] == 1 and s_procstates_1[0] != 2</label>
    <label kind="assignment">c0_proceed[0] = 1, c1_proceed[0] = 0</label>
  </transition>
  <transition>
    <source ref="l0"/>
    <target ref="l0"/>
    <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0</label>
  </transition>
  <transition>
    <source ref="l0"/>
    <target ref="l0"/>
    <label kind="guard">s_procstates_0[0] != 1 and s_procstates_1[0] == 1 and s_procstates_0[0] != 2</label>
    <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 1</label>
  </transition>
  <transition>
    <source ref="l0"/>
    <target ref="l0"/>
    <label kind="guard">s_procstates_0[0] == 1 and s_procstates_1[0] != 2</label>
    <label kind="assignment">c0_proceed[0] = 1, c1_proceed[0] = 0</label>
  </transition>
  <transition>
    <source ref="l0"/>
    <target ref="l0"/>
    <label kind="guard">s_procstates_0[0] != 1 and s_procstates_1[0] == 1 and s_procstates_0[0] != 2</label>
    <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0</label>
  </transition>
  <transition>
    <source ref="l0"/>
    <target ref="l0"/>
    <label kind="guard">s_procstates_0[0] == 1 and s_procstates_1[0] != 2</label>
    <label kind="assignment">c0_proceed[0] = 1, c1_proceed[0] = 0</label>
  </transition>
</template>
<system>c0 = c0_tmpl();

Verified Systems International GmbH
\[ \text{FILE: \textbf{mutex\_2.q}} \]

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

B.3.2 UPPAAL Model: Mutex with 3 Clients

\[ \text{FILE: \textbf{mutex\_3.xml}} \]

```

```xml
<nta>
  <declaration>
    int[0,1] c0_proceed[1] = {0};
    int[0,3] c0_st[1] = {0};
  </declaration>
  <nta>
    <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_procsstates_0[0] = 1, c0_st[0] = 1</label>
    </transition>
    <transition>
      <source ref="start"/>
      <target ref="l2"/>
      <label kind="assignment">s_procsstates_0[0] = 0, c0_st[0] = 0</label>
    </transition>
```

Verified Systems International GmbH
<transition>
  <source ref="10"/>
  <target ref="11"/>
  <label kind="guard">c0_proceed[0]</label>
  <label kind="assignment">s_procstates_0[0] = 2, c0_st[0] = 2</label>
</transition>

<transition>
  <source ref="10"/>
  <target ref="10"/>
  <label kind="guard">!c0_proceed[0]</label>
  <label kind="assignment">s_procstates_0[0] = 1, c0_st[0] = 1</label>
</transition>

<transition>
  <source ref="11"/>
  <target ref="13"/>
  <label kind="assignment">s_procstates_0[0] = 3, c0_st[0] = 3</label>
</transition>

<transition>
  <source ref="11"/>
  <target ref="11"/>
  <label kind="assignment">s_procstates_0[0] = 2, c0_st[0] = 2</label>
</transition>

<transition>
  <source ref="12"/>
  <target ref="10"/>
  <label kind="assignment">s_procstates_0[0] = 1, c0_st[0] = 1</label>
</transition>

<transition>
  <source ref="12"/>
  <target ref="12"/>
  <label kind="assignment">s_procstates_0[0] = 0, c0_st[0] = 0</label>
</transition>

<transition>
  <source ref="13"/>
  <target ref="12"/>
  <label kind="assignment">s_procstates_0[0] = 0, c0_st[0] = 0</label>
</transition>

</template>

<template>
  <name>c1_tmpl</name>
  <location id="start">
    <name>start</name>
    <urgent/>
  </location>
  <location id="10">
    <name>l0</name>
  </location>
  <location id="11">
    <name>l1</name>
  </location>
  <location id="12">
    <name>l2</name>
  </location>
  <location id="13">
    <name>l3</name>
  </location>
</template>
<init ref="start"/>
<transition>
  <source ref="start"/>
  <target ref="10"/>
  <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
</transition>
<transition>
  <source ref="start"/>
  <target ref="12"/>
  <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
</transition>
<transition>
  <source ref="10"/>
  <target ref="11"/>
  <label kind="guard">c1_proceed[0]</label>
  <label kind="assignment">s_procstates_1[0] = 2, c1_st[0] = 2</label>
</transition>
<transition>
  <source ref="11"/>
  <target ref="13"/>
  <label kind="assignment">s_procstates_1[0] = 3, c1_st[0] = 3</label>
</transition>
<transition>
  <source ref="11"/>
  <target ref="11"/>
  <label kind="assignment">s_procstates_1[0] = 2, c1_st[0] = 2</label>
</transition>
<transition>
  <source ref="12"/>
  <target ref="10"/>
  <label kind="assignment">s_procstates_1[0] = 1, c1_st[0] = 1</label>
</transition>
<transition>
  <source ref="12"/>
  <target ref="12"/>
  <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
</transition>
<transition>
  <source ref="13"/>
  <target ref="12"/>
  <label kind="assignment">s_procstates_1[0] = 0, c1_st[0] = 0</label>
</transition>
</template>
<template>
  <name>c2_tmpl</name>
  <location id="start">
    <name>start</name>
    <urgent/>
  </location>
</template>
<transition>
    <source ref="start"/>
    <target ref="10"/>
    <label kind="assignment">s_procstates_2[0] = 1, c2_st[0] = 1</label>
</transition>

<transition>
    <source ref="start"/>
    <target ref="12"/>
    <label kind="assignment">s_procstates_2[0] = 0, c2_st[0] = 0</label>
</transition>

<transition>
    <source ref="10"/>
    <target ref="11"/>
    <label kind="guard">c2_proceed[0]</label>
    <label kind="assignment">s_procstates_2[0] = 2, c2_st[0] = 2</label>
</transition>

<transition>
    <source ref="10"/>
    <target ref="10"/>
    <label kind="guard">!c2_proceed[0]</label>
    <label kind="assignment">s_procstates_2[0] = 1, c2_st[0] = 1</label>
</transition>

<transition>
    <source ref="11"/>
    <target ref="13"/>
    <label kind="assignment">s_procstates_2[0] = 3, c2_st[0] = 3</label>
</transition>

<transition>
    <source ref="11"/>
    <target ref="11"/>
    <label kind="assignment">s_procstates_2[0] = 2, c2_st[0] = 2</label>
</transition>

<transition>
    <source ref="12"/>
    <target ref="10"/>
    <label kind="assignment">s_procstates_2[0] = 1, c2_st[0] = 1</label>
</transition>

<transition>
    <source ref="12"/>
    <target ref="12"/>
    <label kind="assignment">s_procstates_2[0] = 0, c2_st[0] = 0</label>

<transition/>
<source ref="13"/>
<target ref="12"/>
<label kind="assignment">s_procstates_2[0] = 0, c2_st[0] = 0</label>
</transition>
</template>
<template>
    <name>s_tmpl</name>
    <location id="start">
        <name>start</name>
        <urgent/>
    </location>
    <location id="l0">
        <name>l0</name>
    </location>
    <init ref="start"/>
    <transition>
        <source ref="start"/>
        <target ref="l0"/>
        <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>
        <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0, c2_proceed...
        [0] = 1</label>
    </transition>
    <transition>
        <source ref="start"/>
        <target ref="l0"/>
        <label kind="guard">s_procstates_0[0] == 1 and s_procstates_1[0] != 2 and
        s_procstates_2[0] != 2</label>
        <label kind="assignment">c0_proceed[0] = 1, c1_proceed[0] = 0, c2_proceed...
        [0] = 0</label>
    </transition>
    <transition>
        <source ref="start"/>
        <target ref="l0"/>
        <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0, c2_proceed...
        [0] = 0</label>
    </transition>
    <transition>
        <source ref="start"/>
        <target ref="l0"/>
        <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>
        <label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 1, c2_proceed...
        [0] = 0</label>
    </transition>
    <transition>
        <source ref="l0"/>
        <target ref="l0"/>
        <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>
<label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 0, c2_proceed[0] = 1</label>
</transition>

<transition>
<source ref="10"/>
<target ref="10"/>
<label kind="guard">s_procstates_0[0] == 1 and s_procstates_1[0] != 2 and s_procstates_2[0] != 2</label>
<label kind="assignment">c0_proceed[0] = 1, c1_proceed[0] = 0, c2_proceed[0] = 0</label>
</transition>

<transition>
<source ref="10"/>
<target ref="10"/>
<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>
<label kind="assignment">c0_proceed[0] = 0, c1_proceed[0] = 1, c2_proceed[0] = 0</label>
</transition>

</template>

<system>c0 = c0_tmpl();
c1 = c1_tmpl();
c2 = c2_tmpl();
s = s_tmpl();
ystem c0, c1, c2, s;</system>

FILE: mutex_3.q

// GENERATED BY gen-mutex-gtl.awk
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 translation mode ("native", "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` Performing a dry run only
- `-d option` — `--debug=option` Give debugging options (e.g.

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
-D SAFETY

saw option --
Spin Version 6.1.0 -- 4 May 2011
Valid Options are:
-a, -l, -f  -> are disabled by -D SAFETY
-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

Verified Systems International GmbH
−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

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
The value of these variables does not matter. Defining them is 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
−sse3 cx16 xtrm 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
...
<analogous 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