SICS Technical Report
T90/9002

ISRN : SICS-T—90/9002-SE
ISSN : 1100-3154

Identifying some Bottlenecks of the Concurrency Workbench

by

Patrik Ernberg and Lars-åke Fredlund

April 1990

Swedish Institute of Computer Science
Box 1263, S-164 29 KISTA, SWEDEN

Identifying some Bottlenecks of the Concurrency Workbench

Patrik Ernberg and Lars-åke Fredlund*
Swedish Institute of Computer Science
Box 1263, S-16428 Kista, Sweden

April 10, 1990

Abstract

We present results which identify some of the bottlenecks of the Concurrency Workbench (CWB). Our results concentrate on the Minimize command which computes an agent with the smallest state space that is observation equivalent with a supplied agent. Measurements show that three major bottlenecks can be identified and that the performance of the CWB depends heavily on the amount of available primary memory.

1 Introduction

The Concurrency Workbench(CWB) is an automated tool which caters for the analysis of concurrent finite state processes expressed in Milner's Calculus of Communicating Systems(CCS). It has been applied to examples involving the verification of communication protocols and mutual exclusion algorithms and has proved to be a valuable aid in teaching and research.

Recently, industry has taken interest in the Workbench for the verification of concurrent processes. The Swedish Telecom Radio have used the Workbench to verify certain parts of the protocols developed for GSM and Ellemtel have used it to experimentally verify an ISDN protocol. Both these examples were relatively large in nature, requiring Workbench calculations which occasionally took more than a day to complete.

The large computations can to a certain extent be attributed to the lack of a clear methodology when it comes to the verification of concurrent processes. Clearly, abstractions and appropriate modularization can reduce the size of the state space and thereby cut down the size of the computations. By abstraction we mean reducing the level of detail in a specification. However, large abstractions are hard to grasp and often give little confidence to designers developing a large distributed system. Furthermore long computations make an interactive design process between the designer and the Workbench difficult. It is therefore of general interest to identify the bottlenecks in the Workbench in order to focus eventual code optimizations on the most time-consuming functions.

*Authors' email: pernberg@sics.se; fred@sics.se

One Workbench command which has been frequently used in the verification process is the *minimize* command. This command computes an agent with the smallest state space that is observation equivalent with the supplied agent. In the remainder of this paper we will concentrate on this command. The command consists of four major functions: *Mkgraph*,*Obscl*, *Equiv.minimize*, and *MkAgent*. Within Obscl, three other functions can be identified: *Reflexcl*,*Transcl*, and *Actcl*. A more detailed description of these functions will be given in Section 2 of this paper.

Earlier experiments have suggested that the Transcl algorithm implemented in the Workbench is particularly time consuming and could be optimized considerably. An M.Sc. student at SICS has studied other algorithms for the Transcl function and successfully implemented them in the Workbench. Early results seem to suggest that an algorithm by J.Eve and R.Kurki-Suonio [EK77] is the most efficient.

This paper presents results which confirm the efficiency of the the above mentioned algorithm. We also present results related to other functions involved in the minimize command and show how the efficiency of these functions depends on the size of the graphs being minimized as well as on the machine configuration.

In the rest of this paper we will assume that the reader is familiar with CCS and the general architecture of the Concurrency Workbench. We refer to [Mil89] for an extensive explanation of CCS and to [CPS88] and [CPS89] for more detail regarding the architecture and operating instructions of the Workbench.

Section 2 presents the different functions comprised in the minimize command. In section 3 we describe the different tests we performed and the results that were obtained. Lastly, we conclude and give some recommendations regarding further work and optimizations within the Concurrency Workbench.

2 The Minimize Command

The minimize command computes an agent with the smallest state space that is observation equivalent with the supplied agent. The command is divided into 4 major functions:

- *Mkgraph*: Computes a transition graph from a CCS-agent.
- *Obscl*: Is a combination of three graph transformation functions which are appropriate to use with the bisimulation algorithm to prove observation equivalence. The three functions which constitute Obscl are:
  - *transcl*: Replaces $\xrightarrow{\tau}$ with the transitive closure of $\xrightarrow{\tau}$.
  - *actcl*: Adds $(\xrightarrow{\tau^*})\xrightarrow{\alpha}(\xrightarrow{\tau^*})$ to $\xrightarrow{\alpha}$ in a graph which has been produced by Transcl.
  - *reflexcl*: Takes a graph and adds τ-loops to all states.
- *Equiv.minimize*: Takes a graph and returns a "minimal" graph, i.e. a graph where all bisimilar states have been collapsed to single states.
- *MkAgent*: Computes a CCS-agent from the minimal graph produced by Equiv.Minimize

3 Tests

3.1 An Introduction to the Tests

The time measurements reported in this document were performed on various Sun computers including Sparcstations and Sun-3:s. The operating system running on these machines was SUN-OS 4.0.X.

The CWB is written in Standard ML[HMT88] for which a number of different compilers exist. For the measurements we decided to use the New Jersey SML compiler since it

- was available for public use on a large number of platforms including SUN Sparcstations and Sun-3 computers.
- was reasonably efficient in terms of execution speed of the compiled code.
- had features which enabled us to perform precise time measurements.

The timing figures we report are of three kinds: *System time*, *User time* and *Real, elapsed time.* User time is the CPU time consumed while executing instructions in the user space of the CWB. These figures do not include the time the operating system spends servicing system calls requested by the CWB nor the time spent servicing page faults, i.e. fetching missing pages from the swap devices. The User time figures are relatively independent of the amount of memory in the computer that the measurements were run on, except for the (hopefully) small part which is accounted for by garbage collection in user mode. Most of the time spent garbage collecting will be a consequence of waiting for the operating system, since the process of garbage collecting causes a lot of memory faults. The User time figures should therefore accurately reflect the CPU speed of the tested computer. (Unfortunately they will also reflect the quality of the compiler that was used to compile the Concurrency Workbench). Furthermore, User time figures are not affected by the workload on the computer at the time of the test runs.

System time is the CPU time spent by the operating system servicing the CWB. Since the CWB does not do many system calls during the minimization process this figure is relatively unimportant. However, the figures are somewhat affected by paging activities even though the actual time spent waiting for pages from the swap device is not included.

Real, elapsed time is the real amount of time that a user would have to wait during the minimization of an agent. This includes all activities that have to be performed during the minimization including time spent waiting on the virtual memory system (swap device). This figure is naturally heavily affected by the workload, memory configuration and type of swapping device for the computer that the measurements are run on. We have tried to run the tests at times when the workload was low. The figures for Real time can be used to study the impact of different memory configurations on the total time required for minimization of an agent.

3.2 Test 1: Measuring Time Spent by the Different Functions in the Minimize Command

In this section we will identify the functions which account for most of the time spent minimizing an agent with respect to observation equivalence. To be able to reason about

this with some confidence we decided to collect a fairly large set of different CCS agents including for example mutual exclusion algorithms and several different communication protocols.

All tests in this section were performed on a Sparcstation-1 with 16 Megabytes of memory swapping to a local disk. We report results for both User time and Real time. We have done some measurements on the memory usage of the functions involved in the minimization process but will only touch upon the results informally. This topic is also discussed in the next section of the report.

In Table 1 we show the number of states, transitions and τ-transitions in the graphs before the Obscl function, after the Obscl function, and after the completed minimization.

Example	Before Obscl			After Obscl			After Reduction		
	States	Trans.	τ Trans.	States	Trans.	τ Trans.	States	Trans.	τ Trans.
DDM59	59	124	36	59	623	163	31	156	18
LA68	68	135	84	68	752	262	20	80	15
PE78	78	155	109	78	1122	356	29	155	33
UT88	88	185	24	88	379	115	30	80	5
CSMA102	102	243	113	102	4935	1359	9	24	0
NT203	203	416	66	203	1152	299	40	112	5
ABP225	225	589	537	225	14266	5872	2	2	0
GSM275	275	494	474	275	8177	6369	11	32	12
ABP372	372	880	816	372	47741	16061	2	2	0
DI386	386	717	657	386	35326	25343	5	12	2
LA545	545	1582	1378	545	56404	30306	26	212	72
PE725	725	1458	748	725	12715	3240	64	351	51
PRO897	897	1805	1373	897	106111	22110	12	74	19
ABP1011	1011	3027	2897	1011	277553	118177	2	2	0
ABP1446	1446	4699	4506	1446	597516	234101	13	84	24

Table 1: Automata sizes in various stages of the Minimize command

In Figure 1 we display the percentage of total time spent executing in user mode accounted for by the different parts of the minimization process. A graph of the actual time spent executing in user mode is displayed in Figure 2. The agents minimized in the figures are sorted in ascending order with respect to state space size.

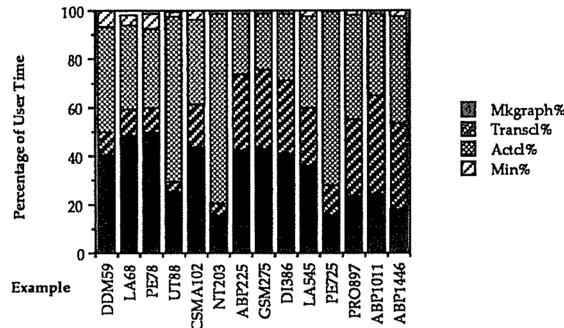

Figure 1: Percentage of User time spent in each function.

As can be seen in the figures, the Actcl, Transcl, Mkgraph and Equiv.minimize functions account for the major part of the execution time. Since the Reflexcl function accounts for a very small amount of the reduction time (less than 1%) it was not included in the figures. The relative importance of the different functions for the total execution

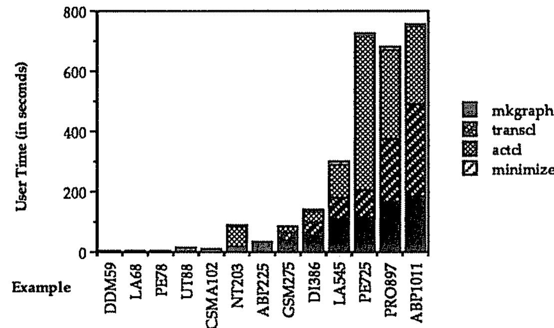

Figure 2: The total User time required for minimization.

time varies a lot for different agents. The measurements of the impact of memory configuration on the real, elapsed execution times show that the Transcl and Actcl functions require lots of memory for their execution, more so than the Mkgraph function. Therefore their importance for the real execution time tends to grow as memory becomes scarce. This is not surprising given what they do (as explained in Section 2): both functions add transitions (edges) to the transition graph constructed by the Mkgraph procedure, thereby requiring more memory for the storage of the graph.

3.2.1 A New Implementation of the Transcl Function

To reduce the time required to minimize agents it was decided to reimplement the Transcl function. An M.Sc. student did an extensive study on transitive closure algorithms [Dah90], implemented some of them, and then studied their impact on the performance of the Concurrency Workbench. As a result of this study, it was decided to use what seemed to be the "most efficient" transitive closure algorithm by J. Eve and R. Kurki-Suonio in a new, experimental version of the Workbench.

We decided to compare the performance of the new Transcl function with the old function. The results of the comparison on the same set of agents as in the preceding section can be seen in Figure 3.

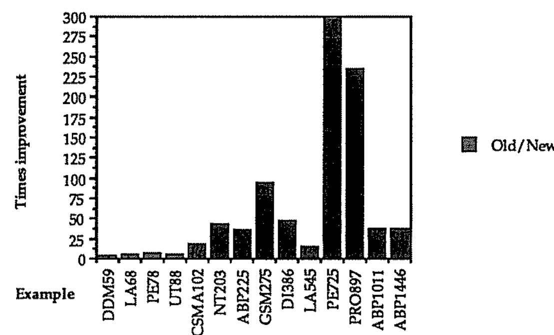

Figure 3: Comparison between the two Transcl functions.

The results show that the gain of using the new algorithm is surprisingly large. The

algorithm is on average 60 times faster than the old one although that figure varies a lot for different agents. The new Transcl function is never less than 5 times faster than the old function. Furthermore, considerably less memory is used by the new Transcl function. The User time percentage figures for a minimization using the new Transcl function is displayed in Figure 4. The real execution times are displayed in Table 2.

Note that a surprisingly large proportion of the Real Time is spent in the Equiv.Minimize function for the ABP1446 example (1135.7 s.). This should be compared with the relatively moderate proportion of User Time spent in the function for the same example (see Figure 4. A closer study of the example revealed that Equiv.Minimize was slow because the amount of primary memory for the storage of the transition graph was too small. As a result, the number of page faults increased catastrophically.

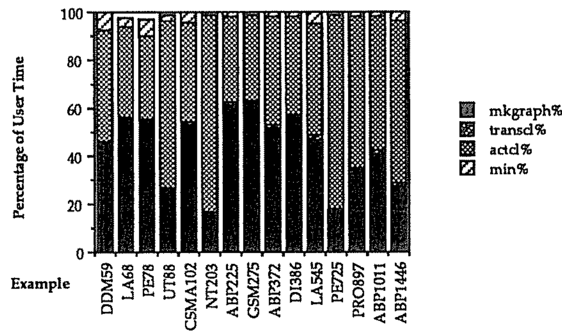

Figure 4: Percentage of User time spent in each function (new Transcl).

In Graph 5 finally, we show the actual User times for a minimization using the old Transcl function compared to the minimization using the new Transcl function. The bars in that graph come in groups of two. The first bar in the group represents a minimization using the old Transcl function, the second the new function.

Examples	Mkgraph	Transcl	Actcl	Minimize	Total
DDM59	2.3	0.0	1.8	0.3	4.5
LA68	2.5	0.0	1.7	0.2	4.5
PE78	3.5	0.0	2.2	0.5	6.3
UT88	3.7	0.0	9.7	0.3	13.9
CSMA102	5.0	0.1	3.7	0.4	9.2
NT203	14.1	0.1	69.3	0.9	84.6
ABP225	15.0	0.3	8.8	0.4	24.5
GSM275	38.5	0.3	21.8	0.8	61.5
ABP372	41.3	0.8	37.3	1.1	80.8
DI386	58.0	1.0	42.2	1.4	103.0
LA545	109.7	4.7	106.5	16.4	238.0
PE725	115.8	0.3	523.0	6.0	645.5
PRO897	166.6	0.9	298.3	11.1	477.5
ABP1011	186.0	15.7	303.3	6.2	512.3
ABP1446	335.7	51.5	2027.5	1135.7	3638.3

Table 2: Real Time in seconds for the examples (new Transcl)

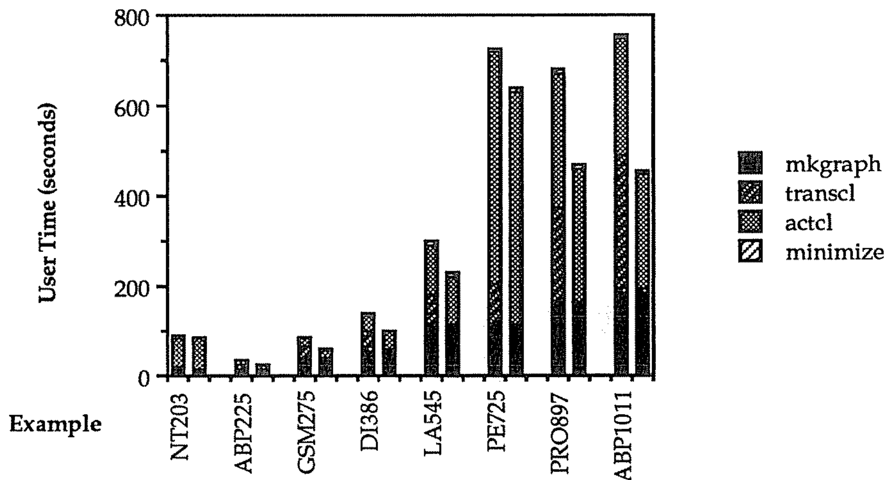

Figure 5: Total User time as influenced by the two Transcl functions.

3.3 Test 2: Time as a Function of Machine Configuration

3.3.1 User Time and Real Time on 16Mb and 8Mb Machines on Cycler Experiment

In this Section we present results obtained when minimizing Milner's scheduling problem (see [Mil80] page 33). This example is interesting for evaluation purposes because the number of states, transitions and equivalence classes grow in the same proportion. It has also been used when testing other tools and algorithms concerned with state space reduction (see, for example, [Fer89]). We have therefore included this example to make comparison with these tools possible. Furthermore, we have performed the same reductions on both an 8Mb and a 16Mb Sparc station in order to evaluate what effect an increase in memory has on reduction times.

The scheduler example has been specified in two different ways. In the first specification, we allow both α and β-actions to be visible actions. In the second specification, we hide β-actions and thus only allow α-actions to be visible. We refer to [Mil80] for a more detailed description and specification of the cycler experiment.

In Table 3 the number of states and transitions after Mkgraph and after Equiv.Minimize for both specifications and for different numbers of cyclers are summarized. The number of transitions that are added by Obscl for both specifications is also given.

	After MkGraph		No. of Trans. After Obscl		After Minimize			
					1st Spec.		2nd Spec.	
Cyclers	States	Trans.	1st Spec.	2nd Spec.	States	Trans.	States	Trans.
2	15	22	56	117	8	12	2	2
3	45	87	198	518	24	48	3	3
4	117	283	592	2009	64	160	4	4
5	285	831	1632	7346	160	480	5	5
6	669	2287	4280	25949	384	1344	6	6
7	1533	9307	10840	89534	896	3584	7	7

Table 3: No. of states and transitions before and after reduction for both specifications

The results using the first and second specifications are presented in Tables 4 and 5. We have included the time taken to perform both Obscl and Equiv.Minimize in order to make comparison with other tools easier. The new Transcl function was used for all the experiments.

Cyclers	Mkgraph	Obscl	Minimize	Obs + Mini	MkAgent	Total
2	0.1	0.1	0.0	0.1	0.0	0.3
3	0.8	0.8	0.1	0.9	0.0	1.8
4	3.8	6.5	0.7	7.2	0.1	11.2
5	18.5	46.6	4.6	51.2	0.5	70.3
6	91.3	290.1	30.7	220.8	2.5	414.7
7	449.5	1793.3	164.9	1958.2	11.5	2419.5

Table 4: User Time in seconds for different functions and cyclers on Sparc Station for first specification

Cyclers	Mkgraph	Obscl	Minimize	Obs+ Mini	MkAgent	Total
2	0.1	0.1	0.0	0.1	0.0	0.3
3	0.7	0.4	0.0	0.4	0.0	1.2
4	3.7	3.3	0.2	3.5	0.0	7.2
5	17.6	23.9	0.5	24.4	0.0	42.0
6	88.5	145.9	1.6	147.5	0.0	236.0
7	436.3	902.0	6.2	908.2	0.0	1344.4

Table 5: User Time in seconds for different functions and cyclers on Sparc Station for second specification

The Concurrency Workbench could not handle more than 7 cyclers. This should be compared to the Aldébaran and Auto tools which can handle 9 and 8 cyclers respectively. Furthermore, these tools seem to have considerably more efficient reduction algorithms. Aldébaran, for example, reduces 7 cyclers for the first specification in 7.3 seconds, i.e several 100 times faster than the Concurrency Workbench. Results on Aldébaran were obtained on a Sun 3/60 with 50 Mb and are therefore not entirely comparable. The results nevertheless remain far superior to those of the Concurrency Workbench.

As a final result, we compare reduction times on an 8Mb Sparc station vs. a 16Mb Sparc station, where $Reductiontime = Obscl + Equiv.Minimize$. The results are illustrated in Table 6 and Figure 6. The experiment was only conducted for the first cycler specification. Note that the scale is logarithmic in Figure 6.

Cyclers	User Time	Real Time(8Mb)	Real Time(16M)
2	0.1	0.1	0.1
3	0.9	1.2	0.9
4	7.4	7.9	7.5
5	51.9	52.2	52.2
6	323.0	426.6	326.3
7	1958.7	2971.8	2009.8

Table 6: User time and Real time in seconds for Reduction of cyclers on 8Mb and 16Mb Sparc Stations for first specification

Again, the sources of error are considerable as we have not performed a fully controlled experiment. The 16Mb station swapped against a local SCSI disk while the 8Mb station swapped against a server on a network. However, the results do seem intuitively correct. We can see that the machines perform equally well on experiments involving 5 cyclers or less. On the other hand, the 16Mb machine is much more efficient on experiments with more than 5 cyclers. A likely explanation is that the heap size becomes larger than the memory available for the 8Mb machine, forcing it to swap and, as a consequence, real time increases considerably.

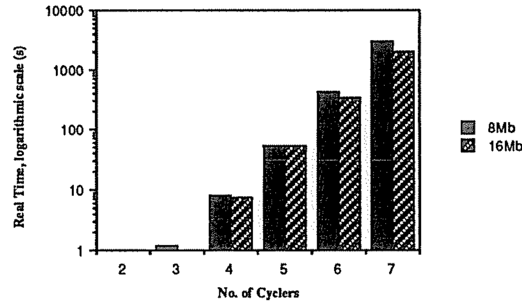

Figure 6: Comparing reduction times for Cyclers on 8Mb and 16Mb Sparc stations.

3.3.2 User Time and Real Time on Different Processors

A small experiment was conducted to compare the efficiency of the workbench on different machines. A Reduction of the communication protocol with 897 states(PRO897 in Section 3.2) was performed on a Sun 3/80, a Sun 4/100, and a Sparc Station each with 8 Mb of memory. The results are presented in Figure 7.

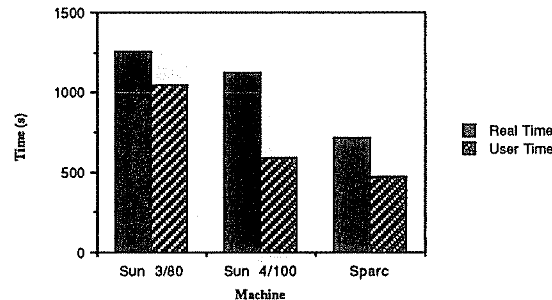

Figure 7: Efficiency of the Concurrency Workbench on Different Machines with 8Mb of memory

The experiment was performed in a network environment where workstations swapped against different servers and is consequently subject to many sources of error. The results merely seem to indicate the not especially sensational fact that Sparc Stations are faster than Sun 4:s which in turn are faster than Sun 3:s.

4 Conclusions and Further Work

In this paper we have presented performance tests on the Concurrency Workbench. Results have shown that the most time and memory consuming functions are MkGraph, Transcl and Actcl. Measurements were performed on Milner's cycler experiment and results were compared to other tools available for the reduction of transition graphs. It turns out that the performance of the Concurrency Workbench is poor in comparison with these tools.

To obtain the best performance on the current version of the Workbench, a fast workstation (eg. Sparc Station) with a fair amount of memory (at least 16Mb) should be used.

Several different strategies can be adopted in order to increase the efficiency of the Workbench. One such strategy is clearly to optimize all functions involved in the minimize command concentrating on those functions which are most time consuming. We have also presented measurements on one such optimization concerning a new Transcl function which turns out to be both faster and less memory-intensive. Presumably similar optimizations could be achieved on all other functions in the minimize command starting with the most time-consuming function Actcl. Optimization of the MkGraph function is also interesting since almost all Workbench commands rely on the construction of a graph from a CCS-agent.

Another strategy is to try to keep the state space small before and during state space reduction. Modifications of the Workbench at Edinburgh [Mor90] also confirm that considerable performance improvements can be achieved using this strategy. Alternatively, a more integrated reduction algorithm which avoids explicitly computing for example the transitive closure of a relation may be a fruitful approach. This has recently been done on a trial implementation of Branching Bisimulation [GV89] with promising results.

Lastly, a reimplementation of the Concurrency Workbench in a more "efficient" language would probably improve its performance considerably. Current ML implementations are quite slow and require a considerable amount of memory.

5 Acknowledgements

We are grateful to all people that have complained loudly about the performance of the CWB. They have inspired us to write this report. We would especially like to thank Joachim Parrow who provided valuable insights into the workings of the CWB and, more importantly, lent us his login directory at Edinburgh. Furthermore we would like to thank the Computer Science Department at the University of Edinburgh for letting us use their machines for the tests.

References

[CPS88] R. Cleaveland, J. Parrow, and B. Steffen. *The Concurrency Workbench: Operating Instructions*, 1988.

[CPS89] R. Cleaveland, J. Parrow, and B. Steffen. A semantics-based verification tool for finite-state systems. In *Protocol Specification, Testing, and Verifiation, IX*, 1989.

[Dah90] M. Dahlberg. Efficient algorithms for computing transitive closure in CWB. Technical Report (To be published), Swedish Institute of Computer Science, Kista, 1990.

[EK77] J. Eve and R. Kurki-Suonio. On computing the transitive closure of a relation. *Acta Informatica*, 8:303–317, 1977.

[Fer89] J-C. Fernandez. Aldébaran: A tool for verification of communicating processes. Technical Report RTC 14, IMAG, Grenoble, 1989.

[GV89] J. Groote and F Vaandrager. An efficient algorithm for branching bisumulation and stuttering equivalence. Technical Report (draft full paper), Centre for Mathematics and Computer Science, Amsterdam, 1989.

[HMT88] R. Harper, R. Milner, and M. Tofte. The definition of Standard ML, version 2. Technical Report ECS-LFCS-88-62, Department of Computer Science, University of Edinburgh, 1988.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, pages 33–36. Springer-Verlag, 1980.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Mor90] M. Morley. New commands for state space reduction. Technical Report (To be published), Department of Computer Science, University of Edinburgh, 1990.