FI MU

Faculty of Informatics
Masaryk University

A Comparison of Algorithms for Normed BPA Processes – An Experimental Performance Evaluation

by

Aleš Borek

FI MU Report Series **FIMU-RS-2001-06**

 September 2001

A comparison of algorithms for normed BPA processes - an experimental performance evaluation

Aleš Borek

September 13, 2001

Abstract

Three algorithms for deciding bisimilarity of normed BPA processes (two using bisimulation base and one using tableau) are evaluated. The evaluation is based on experimental results and some weak and strong points of algorithms with regard to practical use are being shown.

1 Introduction

In this paper we will look at various algorithms for deciding bisimilarity between normed BPA processes and compare them.

The states of a BPA process (see [4] for precious definitions and detailed description) are represented by sequences of symbols from a set Var. The dynamics is described by finite set of rewrite rules of the form $X \stackrel{a}{\rightarrow} \alpha$, where $X \in Var$ is a single symbol, $a \in Act$ is an atomic action and $\alpha \in Var^*$ is a sequence of symbols. The rewriting formalism is prefix-rewriting, which means that the rules are only applied at the leftmost position in the term. For example, a rule $A \stackrel{a}{\rightarrow} BC$ can be applied to a term $ADEF$ and yields the transition $ADEF \stackrel{a}{\rightarrow} BCDEF$.

A norm of process E (norm E) is $\min\{|w| \mid E \stackrel{w}{\rightarrow} \epsilon\}$ where $|w|$ denotes the length of a word w. E is said to be *normed* if norm $E < \infty$.

A relation R on the process set is a *bisimulation* iff whenever $\alpha R \beta$, it holds:

- if $\alpha \stackrel{a}{\rightarrow} \alpha'$ then $\beta \stackrel{a}{\rightarrow} \beta'$ for some β' with $\alpha' R \beta'$; and
- if $\beta \stackrel{a}{\rightarrow} \beta'$ then $\alpha \stackrel{a}{\rightarrow} \alpha'$ for some α' with $\alpha' R \beta'$.

Two processes α and β are *bisimilar* if there exists a bisimulation R such that $\alpha R \beta$.

There are various approaches for deciding if two processes are bisimilar. One of them is based on tableau method, the other one is based on the bisimulation bases.

Tableau algorithm is a goal oriented proof system. A construction starts with an expression $E = F$, where E and F are roots of processes we want to compare. This expression is then divided into a few subgoals by application of proof rules and the same procedure is applied on subtrees. A construction of tableau along some branch is finished when some finishing condition occurs.

The main rules are LCANCEL, which cancels the leftmost parts of processes, SPLIT, which splits the expression into two easier ones, and UNFOLD, which unfolds the leftmost variables and matches resulting expressions to pairs, secondary rules are REDUCE which replaces processes with bisimilar ones (if we have found some during the proof) and SWAP. For more precious description and proofs see [3].

Bisimulation base algorithm is based on iterative decreasing of possible pairs of variables which can be bisimilar. First we have all combination of variables in our bisimulation base. Then we gradually weed out invalid pairs until we finish with a base which is exactly our required relation of bisimulation. To be a bit more precise, a base consists of pairs $(X, Y\alpha)$, where X and Y are variables and α is string (possible empty) of variables. It holds that norm X = norm $Y\alpha$. Base will initially contain one pair $(X, Y\alpha)$ for each distinct variables X and Y. Then we will try iteratively to construct new bases with fewer pairs until there will be no change. We will use very important function $g : Var \to Var^*$ which will be called decomposing function. It will be extended to domain Var^* by defining $g(\epsilon) = \epsilon$ and $g(X\alpha) = g(x)g(\alpha)$. Furthermore we then define $g^*(\alpha)$ for α from Var^* to be the limit $g^t(\alpha)$ as $t \to \infty$. Then for base B, decomposing function g and processes α and β we will define relation $\alpha \equiv_B^g \beta$ as follows:

- if $g^*(\alpha) = g^*(\beta)$ then $\alpha \equiv_B^g \beta$;
- otherwise let X and Y be the leftmost mismatching pair of symbols in the words $g^*(\alpha)$ and $g^*(\beta)$;
  - if there is a γ with $(Y, X\gamma) \in B$ then $\alpha \equiv_B^g \beta$ iff $\alpha \equiv_B^{\hat{g}} \beta$, where $\hat{g} = g[Y \mapsto X\gamma]$;

– otherwise $\alpha \not\equiv^{g}_{B} \beta$

The notation $g[Y \mapsto X\gamma]$ is used to denote function that agrees with g at all points except Y where its value is $X\gamma$. Now we need to get a new base $\hat{B}$. A pair $(X, \alpha) \in B$ will be also included in $\hat{B}$ iff:

- if $X \xrightarrow{a} \beta$ then $\alpha \xrightarrow{a} \gamma$ for some γ with $\beta \equiv^{\mathrm{Id}}_{B} \gamma$; and
- if $\alpha \xrightarrow{a} \gamma$ then $X \xrightarrow{a} \beta$ for some β with $\beta \equiv^{\mathrm{Id}}_{B} \gamma$.

The only difference between exponential and polynomial algorithms is the way relation $\alpha \equiv^{g}_{B} \beta$ is computed. In the case of the exponential algorithm the relation is computed directly by unfolding appropriate variables (i.e. substitution of all X by $Y\gamma$ if $(X, Y\gamma) \in B$) recursively until nothing can be substituted and then resulting strings are compared (note that resulting strings can be quite large). In the case of the polynomial algorithm it is much more difficult. For more precious description and proofs see [9].

2 A comparison of algorithms

In this section we will compare algorithms for normed BPA processes - one algorithm for tableau, one polynomial using bisimulation base and one exponential using bisimulation base.

It is hard to compare these absolutely different algorithms, as the best comparison criterion will be the time we need to decide if processes are bisimulation equivalent (and in lesser extent space). However this is strongly dependent on a concrete implementation which we don't claim is the best one in our case.

Comparisons were ran at AMD Duron 900MHz, 256MB RAM under MS Windows 98. Naturally, no processes ran in background. A comparison of space complexity is mainly for illustration purposes and is not really precise - it was measured as follows: when some space was allocated (`new`, `malloc`) or a local procedure was called, the appropriate number of used space was added (note that we must add space for local variables in procedures) and when corresponding memory was released or a return from procedure occurred, the proper number was substracted - used memory is the maximum number occurring in a run.

There were used several methods for a comparison. Firstly, manually created processes (mainly used for debugging purposes). It is hard to create big scale of bisimilary equivalent processes which don't look so at first

glance and which are sufficiently difficult, so that program will not run too shortly. We decided to generate randomly some process (constrained by constants as the number of variables, K-GNF-form, ...) and compare it to itself. As a result these processes must be bisimilar. This method is not too realistic but on the other hand we are able to compare some features (as a dependence on the number of variables, rules, nondeterminism, ...). Algorithms were in no way optimized to run better when processes look like that (we could for example use this knowledge to make better nondeterministic choices in the tableau algorithm - we will discuss this issue later on). It could be also interesting to have a look at how algorithms behave when processes are not bisimilar - for this case we used former method of a process generation and then some variable was randomly changed to some other - so that processes will not be usually trivially nonbisimilar.

For a basic comparison of algorithms the table 1.[1] may be used.

Each generated process had 20 variables, each variable had 1 - 20 rules, there were altogether 20 possible transitions and processes were in 5-GNF form ($X \stackrel{a}{\rightarrow} \alpha, |\alpha| < 5$). Results are average and maximum values of 200 randomly generated processes.

	Tableau	Base exp.	Base pol.
time [ms]	bisimilar		
average	3	12	348
maximum	11	23	955
memory [kB]			
average	114	111	1124
maximum	141	123	2520
time [ms]	nonbisimilar		
average	4	13	348
maximum	79	25	894
memory [kB]			
average	105	111	1132
maximum	137	127	2644

Table 1: A comparison of algorithms

Very long time for determining bisimulation equivalence when using the most effective algorithm (regarding complexity) could be quite surprising. However, an idea of bases is fairly good, as can be seen by good results

[1]Tableau is Burkart, Caucal, Steffen algorithm, base exp. means a direct comparison of $g^*(\alpha) = g^*(\beta)$

regarding time when we compare $g^*(\alpha) = g^*(\beta)$ directly. Fairly simple tableau algorithm doesn't look bad either. Bigger memory requirements of the polynomial algorithm are there because of the way we precompute all triples (we allocate space for them but don't compute all of them necessarily) - of course everything could be computed "on-the-fly" but it would further decrease time performance. There is no difference in bisimulation base based algorithms when nonbisimilar processes are compared but there can be seen one weakness of the tableau based algorithm - maximum time to decide that processes are really nonbisimilar is a bit longer.

2.1 Tableau algorithm

First, let's have a look at the tableau algorithm. As a base we will take description of generated process listed above. First parameter we can change is the number of variables. In table 2 a dependency on the number of variables is given. *Time* is the length of computation, *Steps* value represent, how many times we visited nodes where one of the rules LCANCEL, SPLIT, UNFOLD was applied. This is fairly important indicator. *Mem* naturally shows the amount of memory used. Average values were counted from 200 generated processes. Maximum values were not much higher than average ones so they are not in table 2 where we compare bisimilar processes. The table for comparing nonbisimilar processes shows only maximum times and steps because there were large discrepancies between various values and so average value would say almost nothing. When comparing processes with 200 hundreds variables one of the comparisons ran over 2 hours and yet did not finish.

Number of variables	20	40	60	80	100	200
	bisimilar					
Time [ms]	4	12	19	26	38	112
Steps	683	1361	2036	2718	3383	6810
Mem [kB]	114	211	307	405	501	992
	nonbisimilar					
Max.time [s]	0.4	33	104	206	2270	-
Max.steps [thousand]	12	4509	8271	19892	205392	-

Table 2: Tableau - changing the number of variables

We can see that the number of steps increases only linearly, a bit worse it is with time increase (but still fairly efficient) as we do not need to ap-

ply rule UNFOLD so often. On the other hand, when processes actually are not bisimilar, results speak that we can expect fairly (exponentially) high times. In most cases the time needed to reach the result is small - for example when we compare processes with 40 variables and compute an average time needed for a comparison of 100 generated processes we will get a value 400 ms but when we compute an average of 96 smallest values we will get a value 19 ms (still a bit higher than when processes are bisimilar but not much). But those problematic 4 processes need a long time to correctly decide that they are not bisimilar.

Another parameter we can change is the number of rules on the right side. Results are in table 3.

Number of rules	1–20	10–29	20–39	30–49	100–119
	bisimilar				
Time [ms]	5	8	17	25	5337
Steps	683	1321	2065	2870	66152
	nonbisimilar				
Time [ms]	8	26	86	695	-
Steps	798	3568	11613	80740	-

Table 3: Tableau - changing the number of rules

Although it might look that time and step increases are almost linear when processes are bisimilar, it is not the case as can be seen if the number of possible rules is large (100 in our case).

In table 3 we provide average values for comparing nonbisimilar processes rather than maximum ones as these are not much higher (they were 20 to 50 times higher than average ones - the worst one was 30 seconds). It means that increasing the number of rules in a reasonable scale is not as bad for the tableau algorithm. On the other hand when we increase the number of rules considerably, achieved times are much worse than those when the number of variables was increased. There is much higher probability that a comparison of nonbisimilar processes with 100 rules will run over one hour than that when comparing processes with 200 variables and 20 rules as in previous case (see table 2).

Another table 4 shows performance when changing K value at K-GNF form. It goes only to 20-GNF because it is hard to create random normed processes with K-value around 100. Also it seems to be quite unrealistic to consider such processes.

The performance of the tableau algorithm is quite good in this case as

K-GNF	5	8	10	13	20
	bisimilar				
Time [ms]	5	10	15	26	99
Steps	683	1315	1736	2374	4063

Table 4: Tableau - changing K-GNF

well. There is no table for results achieved when comparing nonbisimilar processes. Again, on average, values would not be much higher, but a few generated processes (like one case out of 50) achieve really high times which was not the case in examining previous cases. The number of steps increases rapidly with longer time in this case.

Another problem (in general for a tableau method) is an increase of exponentiality when we increase nondeterminism. We can achieve this by decreasing the number of transitions - see table 5.

Number of actions	1	3	5	20	60
	bisimilar				
Time (ms)	-	27	6	6	6
Steps	-	935	753	683	671

Table 5: Tableau - changing the number of transitions

It is almost impossible to compare processes which have only one possible action using this algorithm (running times are too long). Nevertheless, it is a bit different problem - and the tableau algorithm is not suited for it. Of course a comparison of processes with high number of possible transitions is easier for this algorithm - there is almost no nondeterminism there and that is where the tableau algorithm shines (in our case with 60 transitions the achieved results are almost similar to case with 5 transitions because the number of rules is 1–20 for one variable and there is fairly small nondeterminism in both cases). Values measured for time can differ a few milliseconds - the only notable increase is when we have 3 possible transitions and time is proportionally higher to the number of steps. That is an interesting thing in this case, that when decreasing the number of possible transitions, high time does not always mean high number of steps unlike in previous cases.

2.2 Bisimulation base algorithm

When comparing algorithms using bisimulation base we cannot use something similar to the number of steps. Since the exponential algorithm is a simplification of the polynomial one, there is no possibility how to compare them other than measuring time (or memory) of run. For a comparison with the previous algorithm we will use the number of calling of procedure for comparing of bases (which can be similar to the number of *steps* parameter in the previous algorithm).

Bisimulation base algorithms are fairly weak when we increase the number of variables. It is mainly noticeable in the case of the polynomial algorithm where the running time of 200 variables comparison is exceedingly high. The details can be seen in table 6. *Exp.Time* and *Pol.Time* denote time of one run of the exponential (or polynomial respectively) algorithm. Accordingly, *Exp.Mem* and *Pol.Mem* denote an amount of used memory. Last value, *comparison of base*, means the number of the procedure invocations used for the base comparison (in other words, how many times we needed to know if $\alpha \equiv_B^g \beta$ for some α and β).

Another problem is an increase of memory requirements for the polynomial algorithm - in the worst case all triples must be computed and it is much effective to store them somewhere. Of course in the case that everything necessary is computed "on-the-fly" it is possible to decrease memory requirements but a computation would last much longer.

Number of variables	20	40	60	80	100	200
	bisimilar					
Exp.Time [ms]	12	152	778	5607	15524	300355
Pol.Time [s]	0.3	7.5	54	219	654	-
Exp.Mem. [kB]	111	317	628	1046	1576	5790
Pol.Mem. [MB]	1.1	7.8	25	58	108	-
comparison of bases		5030	9697	15880	27241	86163
	nonbisimilar					
Exp.Time [ms]	12	152	761	5705	15357	307299
Pol.Time [s]	0.3	7.2	55.6	180	533	-
Exp.Mem. [kB]	111	317	631	1049	1568	5789
Pol.Mem. [MB]	1.1	7.9	24.6	54.9	109	-
comparison of bases	1832	5148	11260	16752	23783	86890

Table 6: Bisimulation base - changing the number of variables

The number of the procedure invocations for the base comparison (*com-*

parison of bases) doesn't increase so much. When using the polynomial algorithm it can be also seen that stalling in this procedure is strongly dependent on the number of variables. Actually not even one comparison of processes with 200 variables did finish (it would probably take 3.2GB of memory and ran for a really long time).

Another feature of bisimulation base algorithms is that there is no significant difference between comparing nonbisimilar and bisimilar processes (in time, space and the number of steps). It is important to note that the comparison was not stopped when we realized that root variables are not in a bisimulation base but only when the whole correct bisimulation base was created. It would be possible to further improve an average speed of algorithm if we are not interested in an inner structure of processes.

When comparing tableau and bisimulation base algorithms concerning the number of variables it can be seen that the maximum time for comparing nonbisimilar processes when using tableau is roughly the same as when comparing processes using polynomial base algorithm. For example, when we have 40 variable processes and tableau based algorithm, only one run out of 100 was over 30 seconds, 2 others were over 0.5 seconds and 6 others were over 0.1 seconds. On the other hand there were 8 trivial comparisons which lasted less than 1 microsecond.

When increasing the number of rules (see table 7), algorithms start to achieve good results, mainly the exponential one where time of computation increases more slowly than when using the tableau algorithm (see table 3).

Number of rules	1–20	10–29	20–39	30–49	100–119
Exp.Time [ms]	12	21	33	48	213
Pol.Time [ms]	354	548	841	1183	4319
comparison of bases	1791	3493	5796	8427	38095

Table 7: Bisimulation base - changing the number of rules

The time increase when stepping from 30–49 to 100–119 is much smaller than when using the tableau algorithm. It is even strictly better to use the polynomial time algorithm when comparing processes with over 100 rules where the tableau algorithm "breaks". Even when having 30–49 rules an exponential one is only twice slower in an average than the tableau algorithm when comparing bisimilar processes and of course much better when processes are nonbisimilar.

Another good results will be achieved when increasing K (K-GNF form).

There is no increase in the number of procedure invocations for comparing bases as can be seen in table 8. There is fairly high oscillation of times when using the polynomial algorithm in this case, which is the reason maximum times were added to the table. It is the only case where this problem actually occurs - the cause is that strings $g^*(\alpha)$ and $g^*(\beta)$ are a bit longer than in previous cases and thus it takes longer time to compare them. The polynomial time algorithm is fairly bad in this case - we cannot easily predict time necessary to finish and it is slow.

K-GNF	5	8	10	13	20
Exp.Time [ms]	12	29	42	71	231
Pol.Time [ms]	354	948	1896	4928	14842
Pol.Max.Time	957	5244	13778	18238	71243
Pol.Mem [MB]	1.1	1.9	3.0	5.6	12.5
Pol.Max.Mem [MB]	2.5	9.0	14.7	22.8	37.3
comparison of bases	1791	1823	1840	1865	1877

Table 8: Bisimulation base - changing K-GNF

A common advantage of bisimulation base based algorithms can be also seen when processes are highly nondeterministic. When they have only one transition action, the number of callings of the procedure for comparing bases sharply increases, but the two algorithms finish in feasible time (see table 9).

Number of actions	1	3	5	20	60
Time (ms) (exp.)	405	76	43	12	7
Time (ms) (pol.)	8483	1668	957	344	234
comparison of bases	44017	8616	5060	1791	1146

Table 9: Bisimulation base - changing the number of transition actions

It is even possible to compare processes with one action which was impossible in the previous algorithm. But perhaps surprisingly the tableau algorithm is not as ineffective when we have only 3 possible actions, it is still thrice faster than the polynomial base algorithm when processes are bisimilar. Another noticeable thing which occurs in this case is some time improvement when increasing the number of possible transitions from 5 to 60 (where the processes are almost deterministic) - in the tableau algorithm the number of steps almost didn't decrease and time was the same. It means that when processes are almost deterministic (60 transitions and

only 20 rules) both tableau and exponential bisimulation base based algorithms take 6–7 ms. But when processes are only slightly nondeterministic and relatively small (5 transitions and 20 rules) then tableau based algorithms takes still 6 ms unlike other one which now takes 43 ms. But when we proportionally increase the size of problem (20 transitions and 100 rules) the tableau algorithm starts to break and achieves worse results.

3 Conclusion

Our experiments demonstrate that the exponential algorithm using bisimulation base is better than its polynomial version. If the base would be like $(X, YYYY)$, $(Y, ZZZZ)$, ..., it could create a huge word which could be impossible to develop directly. The only algorithm which could succeed would be the polynomial one. Of course, its run would last really long, and we conjecture that there is almost no chance to find these kind of processes in practice.

The tableau based algorithm has one problem which is as follows: comparing some nonbisimilar processes can take huge amount of time (usually one process out of 100). In average it is better than the other algorithms for everything save processes with only one action and processes with over 100 rules. The good thing is that when the time is really high so is the number of steps so that a computation can be interrupted and some other algorithm can be used. It behaves especially well in a comparison with other algorithms when the number of variables is high (with bisimilar processes it is thousand times faster) but again few processes really decrease the performance and necessary time for comparing some of them goes over 2 hours. On the other hand sometimes finding that processes are nonbisimilar can take really small time and is almost instantaneous.

It could be further possible to improve times when comparing bisimilar processes by using some heuristics in our case. For example when we have processes $A \rightarrow aB$ ($A \stackrel{a}{\rightarrow} B$) and $A' \rightarrow aB'C' + aD'$ we will first try to explore trees B and D' because the number of variables on the right-hand side starting with an action a is the same too unlike with rule $aB'C'$. But this could be useful only if bisimilar processes we want to compare have basically the same inner structure and it would not help with nonbisimilar processes at all.

It is possible to improve the polynomial algorithm but we will never reach good times appearing when using tableau algorithms for smaller processes. The best solution is probably to merge advantages of both al-

gorithms. When the nondeterminism is high or rules are long, it is better to use the bisimulation base algorithm, when a lot of variables appears the tableau algorithm would be chosen. There is of course one problem, i.e. we can hit some nonbisimilar processes which take a lot of time using the tableau algorithm and it would be advantageous to stop its computation after a while and use the bisimulation base algorithm instead. The problem is where exactly should be put this boundary (it probably has to be based on experimental results). Another possibility how to merge these algorithms is to use a few steps of tableau method when putting some variables to bisimulation base for the first time; we can quickly realize that they are not bisimilar and not include them.

The exponential algorithm using bisimulation bases is easiest to implement, which is nice because of its running times. On the other hand, the polynomial algorithm using bisimulation base is hardest to implement - mainly because lots of possibilities when comparing $g^*(\alpha)$ with $g^*(\beta)$. It is difficult to check them all.

References

[1] A. Borek. On bisimulation of normed and unnormed BPA processes (in Czech), Master Thesis, Masaryk University, Brno, 2000.

[2] O. Burkart. Automatic Verification of Sequential Infinite-State Processes, *Lecture Notes in Computer Science*, Vol.1354, Springer Verlag, 1997.

[3] O. Burkart, D. Caucal, B. Steffen. An Elementary Bisimulation Decision Procedure for Arbitrary Contex-Free Processes, AAchener Informatikbericht Nr. 94-28, 1994.

[4] O. Burkart, D. Caucal, F. Moller, B. Steffen. Verification on infinite structures, Handbook of Process Algebra (Edited by J.A. Bergstra, A. Ponse, S.A. Smolka), Elsevier, 2001.

[5] S. Christensen, H. Hüttel, C. Stirling. Bisimulation Equivalence Is Decidable for All Context-Free Processes, pp.138-147, *Lecture Notes in Computer Science*, Vol.630, Springer, 1992.

[6] R. J. van Glabbeek. The Linear Time - Branching Time Spectrum, pp.278-297, *Proceedings of CONCUR'90, Lecture Notes in Computer Science*, Vol.458, Springer, 1990.

[7] J. F. Groote, H. Hüttel. Undecidable Equivalences for Basic Process Algebra, pp.354-371, *Information and Computation*, 115(2), 1994.

[8] Y. Hirshfeld, M. Jerrum, F. Moller. A polynomial algorithm for deciding bisimilarity of normed contex-free processes, pp.143-159 *Theoretical Computer Science*, Vol.158, 1996.

[9] Y. Hirshfeld, M. Jerrum, F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes, pp.251-259 *Mathematical Structures in Computer Science*, Vol.6, 1996.

[10] H. Hüttel, C. Stirling. Actions Speak Louder than Words: Proving bisimilarity for Contex-Free Processes, pp.485-509 *Journal of Logic Computation*, Vol.8, No.4, 1998.

Publications in the FI MU Report Series are in general accessible via WWW and anonymous FTP:

`http://www.fi.muni.cz/informatics/reports/`
`ftp ftp.fi.muni.cz (cd pub/reports)`

Copies may be also obtained by contacting:

**Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic**