Exploring the Scope for Partial Order Reduction

Jaco Geldenhuys[1], Henri Hansen[2], and Antti Valmari[2]

[1] Computer Science Division, Department of Mathematical Sciences
Stellenbosch University, Private Bag X1, 7602 Matieland, South Africa
jaco@cs.sun.ac.za
[2] Department of Software Systems, Tampere University of Technology
PO Box 553, FI-33101 Tampere, Finland
{henri.hansen,antti.valmari}@tut.fi

Abstract. Partial order reduction methods combat state explosion by exploring only a part of the full state space. In each state a subset of enabled transitions is selected using well-established criteria. Typically such criteria are based on an upper approximation of dependencies between transitions. An additional heuristic is needed to ensure that currently disabled transitions stay disabled in the discarded execution paths. Usually rather coarse approximations and heuristics have been used, together with fast, simple algorithms that do not fully exploit the information available. More powerful approximations, heuristics, and algorithms had been suggested early on, but little is known whether their use pays off. We approach this question, not by trying alternative methods, but by investigating how much room the popular methods leave for better reduction. We do this via a series of experiments that mimic the ultimate reduction obtainable under certain conditions.

1 Introduction

Partial order reduction is a widely-used and particularly effective approach to combat the state explosion problem. Broadly speaking, partial order reduction rules out a part of the state space as unnecessary for verifying a given property. This is achieved by exploiting the commutativity of transitions that arises from their concurrent execution or other reasons. The term "partial order reduction" is somewhat inaccurate, but it is used for historical reasons.

There are several approaches to partial order reduction. For the purpose of this paper, we consider methods that for each state expand some subset of enabled transitions. Such methods are highly similar; the sets of transitions that are expanded are called either stubborn sets [12], ample sets [11], or persistent sets [4]. The methods differ slightly in the way they are defined, and each method has a number of different formulations. Nonetheless, the key ideas in all of them are more or less equivalent [13].

In this paper we use the ample set method as presented in [1] as the starting point of our investigation. It is easy to implement, and its primitive operations are fast but, in return, it wastes some reduction power. We investigate experimentally how much potential there is for better reduction. We are interested in

Z. Liu and A.P. Ravn (Eds.): ATVA 2009, LNCS 5799, pp. 39–53, 2009.

the ultimate results that would be obtainable by an ideal method that is based on partial order principles.

We restrict our attention to reductions that preserve deadlocks. More sophisticated verification questions use additional rules such as visibility of transitions and various cycle closing conditions. We postpone them for future work, because they introduce more complications than can be discussed in this paper.

The concept of *dependency* between transitions refers to situations where two transitions may interfere with each other. Dependency is central in the calculation of ample sets, and some approximation that overapproximates dependency is used. We explore how much additional reduction in the resulting state space is to be gained from using a more accurate dependency relation. Analysis of dependency could be taken further by engaging in *dynamic analysis* as in [3,10], where the notion of dependency is refined during the generation of state spaces.

There is even more variability in the treatment of disabled transitions that are dependent on transitions in the ample set. The correctness of the methods requires that they remain disabled until a transition in the ample set occurs. This can be ensured using different heuristics, some more complicated and presumably also more powerful than others. In this paper we use a rather straightforward *precedence* heuristic and leave the allegedly stronger ones for future work.

In addition to the issues above, there is the question of calculating ample sets. The basic algorithm considers only subsets that are local to a single process. If no suitable process is found, it gives up and returns the set of all enabled transitions. However, a more sophisticated algorithm can often do better in this situation. How much further reduction is gained from using the more sophisticated algorithm, is also a matter of investigation here.

Section 2 defines the formal model of concurrent systems that is used throughout the paper. Section 3 describes exactly how ample sets and the transition dependency and precedence we employ are calculated. In Section 4 we compare experimental results from using the different algorithms for ample sets and different versions of dependencies and precedences, and conclusions are presented in Section 5.

2 Mathematical Background

In the first part of this section we present a formal description of our model of computation. This is not mere formality for its own sake. Although the formalization is detailed, its purpose is to make it possible to describe the computation of ample sets in a precise manner.

2.1 Model of Computation

Our model of computation has three components: a set of variables, a set of transitions, and a set of processes.

Definition 1. *Let $V = (v_1, v_2, \ldots, v_n)$ be an ordered set of variables. The values of variable v_i are taken from some finite domain X_i. Let $X = X_1 \times X_2 \times \cdots \times X_n$.*

- *An* evaluation $e \in X$ *associates a value with each variable.*
- *A* guard $g\colon X \to B$ *is a total function that maps each evaluation to a Boolean value.* $B = \{\mathsf{true}, \mathsf{false}\}$.
- *An* assignment $a\colon X \to X$ *is a total function that maps one evaluation to another.* □

Each process is described fully by its transitions and a designated variable called a *program counter*. A transition is a pair of the form (g, a) where g is a guard and a is an assignment. A transition is enabled in states where its guard evaluates as true. If the transition fires (i.e., is executed), it changes the values of variables as described by its assignment component. In almost all cases, the guard checks that the program counter has an appropriate value, and the assignment changes the value of the program counter.

Definition 2. *A* process *over variables* V *is a pair* (Σ, pc)*, where* Σ *is a set of transitions, and* $pc \in V$ *is a variable called the* program counter*. For each transition* (g, a) *in* Σ *there is a value* x *such that the guard* g *is of the form* $(pc = x) \wedge \varphi$. □

For $e \in X$, we write $pc(e)$ to denote the value of the program counter at e.

Definition 3. *Let* V *be an ordered set of variables over the finite domain* X *and let* $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ *be a set of processes over* V*, such that* $P_i = (\Sigma_i, pc_i)$*. We assume all the* pc_i *are different. Let* $\hat{e}$ *be an evaluation. The* state space *of* $\mathcal{P}$ *from* $\hat{e}$ *is* $M = (S, \hat{e}, \Sigma, \Delta)$*, where* $\hat{e} \in X$ *is the* initial state*,* $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_k$*, and* S *and* Δ *are the smallest sets such that* $S \subseteq X$*,* $\Delta \subseteq S \times \Sigma \times S$*,* $\hat{e} \in S$*, and if* $t = (g, a) \in \Sigma$ *and* $s \in S$ *and* $g(s) = \mathsf{true}$*, then* $a(s) \in S$ *and* $(s, t, a(s)) \in \Delta$. □

We say that S is the set of *states* and $\hat{e}$ is the *initial state*. The elements of Σ above are referred to as *structural transitions*, while the elements of Δ are called *semantic transitions*. From now on, when we write just "transition", we are referring to the structural ones. This convention agrees with Petri net terminology and disagrees with process algebra and Kripke structure terminology.

The following notation is used throughout the paper:

Definition 4. *Let* $M = (S, \hat{e}, \Sigma, \Delta)$ *be the state space of* $\mathcal{P}$ *from* $\hat{e}$*, and let* s *be some state of* M*.*

- *The* enabled transitions *of* s *are* $en(s) = \{t \in \Sigma \mid \exists s' \in S\colon (s, t, s') \in \Delta\}$.
- *We write* $s \xrightarrow{t}$ *when* $t \in en(s)$.
- *We write* $s \xrightarrow{t} s'$ *when* $(s, t, s') \in \Delta$.
- *We write* $s \xrightarrow{t_1 t_2 t_3 \cdots}$*, when* $\exists s_1, s_2, \ldots \in S : s \xrightarrow{t1} s_1 \xrightarrow{t2} s_2 \xrightarrow{t3} \ldots$
- *The* local transitions *of process* P_i *in state* s *are* $current_i(s) = \{(g, a) \in \Sigma_i \mid \exists e \in X : pc_i(s) = pc_i(e) \wedge g(e) = \mathsf{true}\}$.
- *The* enabled local transitions *of* P_i *are* $en_i(s) = current_i(s) \cap en(s)$.
- *The* disabled local transitions *of* P_i *are* $dis_i(s) = current_i(s) \setminus en(s)$. □

Lastly, we need to know which other variables are involved in which transitions. These relationships are only defined in the context of a state space.

Definition 5. *Let $V = (v_1, v_2, \ldots, v_n)$ be an ordered set of variables, let $\mathcal{P}$ be a set of processes over V, and let $M = (S, \hat{e}, \Sigma, \Delta)$ be the state space of $\mathcal{P}$ from $\hat{e}$. Given evaluations $e, e' \in X$, for $1 \leq i \leq n$:*

- *e_i denotes the value of v_i in e, and*
- *$\delta(e, e') = \{i \mid e_i \neq e'_i\}$ is the set of indices on which e and e' disagree.*

If $t = (g, a) \in \Sigma$, then

- *the* test set *of t is $Ts(t) = \{v_i \mid \exists e, e' \in X : \delta(e, e') = \{i\} \wedge g(e) \neq g(e')\}$,*
- *the* write set *of t is $Wr(t) = \{v_i \mid \exists e \in X : e_i \neq a(e)_i\}$,*
- *the* read set *of t is $Rd(t) = \{v_i \mid \exists e, e' \in X : \delta(e, e') = \{i\} \wedge \exists j : v_j \in Wr(t) \wedge a(e)_j \neq a(e')_j\}$, and*
- *the* variable set *of t is $Vr(t) = Ts(t) \cup Rd(t) \cup Wr(t)$.* □

The test, write, and read sets are conservative syntactic estimates of those variables that may be involved in the different aspects of a transition's execution. Variables with disjoint variable sets are clearly independent, but an even finer condition will be given in Section 3.1..

2.2 Ample Sets, Dependency and Precedence

It is important to distinguish which transitions may interfere with one another and to this end we define the following:

Definition 6. *Let $M = (S, \hat{e}, \Sigma, \Delta)$ be the state space of $\mathcal{P}$ from $\hat{e}$, and $S' \subseteq S$. A* dependency relation *$D \subseteq \Sigma \times \Sigma$ for S' is a symmetric, reflexive relation such that $(t_1, t_2) \notin D$ implies that for all states $s \in S'$, the following are true:*

1. *If $s \xrightarrow{t_1} s'$ and $s \xrightarrow{t_2}$, then $s' \xrightarrow{t_2}$ (independent transitions do not disable one another).*
2. *If $s \xrightarrow{t_1 t_2} s'$ and $s \xrightarrow{t_2 t_1} s''$ then $s' = s''$ (the final state is independent of the transition order).*

Given D and $T \subseteq \Sigma$, we write $dep(T) = \{t \mid \exists t' \in T : (t, t') \in D\}$. □

The definition implies that if D is a dependency relation, D' is symmetric, and $D \subseteq D'$, then D' is also a dependency relation. If $S' = S$ in this definition, we get the usual notion of dependency. We shall, however, also use other S'.

Here we use the definition of ample sets from [1], modified to accommodate S' instead of S.

Definition 7. *Let $M = (S, \hat{e}, \Sigma, \Delta)$ be the state space of $\mathcal{P}$ from $\hat{e}$, let $S' \subseteq S$ and let D be a dependency relation for S'. A set $ample(s) \subseteq \Sigma$ of transitions is an* ample set *for state $s \in S'$ if and only if the following hold:*

A0 $ample(s) = \emptyset$ if and only if $en(s) = \emptyset$.
A1 For every path of the state space that begins at s, no transition that is not in $ample(s)$ and is dependent on a transition in $ample(s)$, can occur without some transition in $ample(s)$ occurring first.

Because condition A1 talks about future transitions, some of which may not be enabled in the current state, we need information about not only those transitions that are currently enabled, but also those that may become enabled in the future.

Definition 8. *Let $M = (S, \hat{e}, \Sigma, \Delta)$ be the state space of $\mathcal{P}$ from $\hat{e}$, and $S' \subseteq S$. A* precedence relation $R \subseteq \Sigma \times \Sigma$ *for S' is such that if there is some state $s \in S'$ such that $\neg s \xrightarrow{t_2}$ and $s \xrightarrow{t_1 t_2}$, then $(t_1, t_2) \in R$.*

Given R and $T \subseteq \Sigma$, we write $pre(T) = \{t \mid \exists t' \in T : (t, t') \in R\}$. □

The definition implies that if R is a precedence relation and $R \subseteq R'$, then R' is also a precedence relation. The precedence relation makes it possible to detect all the transitions that can enable a given transition. It is a coarse heuristic. A finer heuristic, commonly used in the stubborn set method, takes advantage of the fact that if the guard of t is of the form $\varphi \wedge \psi$ where φ evaluates to true and ψ evaluates to false in the current state, then it would suffice to only consider those transitions which may affect ψ.

3 The Calculation of D, R, and Ample Sets

3.1 Calculating D and R

In Section 2.1 we carefully make a distinction between structural and semantic transitions, and between the description of a model (its variables and processes) and the state space it generates. This difference plays an important role when it comes to the calculation of the D and R relations.

Specifically, the static (structural) description of a model is almost always available and can be used to calculate an overapproximation of the smallest possible D and R. The full state space, on the other hand, is seldom available for realistic models; the purpose of partial order methods is to avoid its construction!

In this paper we consider three versions of D/R. The first, which we refer to as D_s/R_s, is based on the static model description and is calculated as follows: Let $M = (S, \hat{e}, \Sigma, \Delta)$ be a state space, and $t_1, t_2 \in \Sigma$.

- $(t_1, t_2) \in D_s$ if and only if $Wr(t_1) \cap Vr(t_2) \neq \emptyset$ or $Wr(t_2) \cap Vr(t_1) \neq \emptyset$.
- $(t_1, t_2) \in R_s$ if and only if $Wr(t_1) \cap Ts(t_2) \neq \emptyset$.

It is easy to see that these dependency and precedence relations are not the smallest possible. For example, consider the two transitions

$$t_a : \quad \mathsf{true} \longrightarrow x := (x + 1) \bmod 5$$
$$t_b : \quad \mathsf{true} \longrightarrow x := (x + 2) \bmod 5$$

(Here we use Dijkstra's guarded command notation to describe the guards and assignments of the transitions.) If t_a and t_b belong to two different processes, they are clearly independent: they cannot disable each other and the order in which they execute has no effect on the final state. Nevertheless, according to

the rules above $(t_a, t_b) \in D_s$ because each transition reads a variable (x) that is written to by the other.

The second version of the relations is called D_f/R_f, and is based on the full state space:

- $(t_1, t_2) \in D_f$ if and only if for some state $s \in S$ where $s \xrightarrow{t_1} s_1$ and $s \xrightarrow{t_2} s_2$, either $\neg s_1 \xrightarrow{t_2}$ or $s_1 \xrightarrow{t_2} s' \wedge \neg s_2 \xrightarrow{t_1} s'$.
- $(t_1, t_2) \in R_f$ if and only if for some state $s \in S$, both $\neg s \xrightarrow{t_2}$ and $s \xrightarrow{t_1 t_2}$.

The third and final version of the relations, D_d/R_d, is based on the full state space and the current state:

- $(t_1, t_2) \in D_d(s)$ if and only if for some state $s' \in S$ reachable from s and where $s' \xrightarrow{t_1} s_1$ and $s' \xrightarrow{t_2} s_2$, either $\neg s_1 \xrightarrow{t_2}$ or $s_1 \xrightarrow{t_2} s'' \wedge \neg s_2 \xrightarrow{t_1} s''$.
- $(t_1, t_2) \in R_d(s)$ if and only if for some state $s' \in S$ reachable from s, both $\neg s' \xrightarrow{t_2}$ and $s' \xrightarrow{t_1 t_2}$.

Note that the definition of ample sets in Definition 7 is only sensitive to what happens in the current state and its subsequent states. It is therefore correct to restrict D_d and R_d to the part of the state space that is reachable from the current state. Here we make use of the S' in Definitions 6 and 8: for $D_d(s)$ and $R_d(s)$, we let S' be the set of all those states that are reachable from s.

D_s/R_s are based on structural transitions, whereas both D_d/R_d and D_f/R_f are defined with respect to semantic transitions. While in practice the latter two versions may be expensive to calculate in full, they provide some idea of the limits of partial order reduction.

3.2 Calculating Ample Sets

It is reasonable to always consider all the current transitions in a given process as dependent, and therefore the smallest possible sets that are eligible as ample sets are the sets $en_i(s)$ of enabled local transitions in each process. A conservative estimate of when such a set can be selected is based on the following sufficient condition [1]:

Proposition 1. *$en_i(s)$ is an ample set if for each process $P_j \neq P_i$ we have*

1. *$pre(dis_i(s)) \cap \Sigma_j = \emptyset$, and*
2. *$dep(en_i(s)) \cap \Sigma_j = \emptyset$.* □

A straightforward method for using this information tests the en_i sets one by one. If either of the conditions fails to hold, the set is discarded and we consider the next candidate. If no suitable candidate is found, the set $en(s)$ is used as an ample set. This approach is shown in Figure 1, and it is also roughly how partial order reduction is implemented in SPIN [7].

As it stands, the algorithm returns the first valid ample set it encounters. This is somewhat arbitrary, since it depends on the order in which the processes are examined, which, in turn, often depends on their order of declaration. This may

```
AMPLE1(S)
1 for i ∈ {1, ..., k} such that en_i(s) ≠ ∅ do
2     A ← true
3     for j ≠ i do
4         if pre(dis_i(s)) ∩ Σ_j ≠ ∅ or dep(en_i(s)) ∩ Σ_j ≠ ∅ then
5             A ← false
6             break
7     if A then return en_i(s)
8 return en(s)
```

Fig. 1. Ample set selection from [2]

give the user some control over the selection of ample sets, but it is doubtful whether such control is ever exercised and whether it is effective. Instead, we refer to this default version in Figure 1 as *first choice*, and we consider two alternative approaches:

- *Minimum choice:* The algorithm is modified to compute all the valid ample sets of the form $en_i(s)$ (it merely records the set index in line 7), and returns the smallest set (or one of the smallest sets) in line 8, reverting to $en(s)$ if no $en_i(s)$ qualifies.
- *Random choice:* As for minimum choice, the algorithm computes all valid ample sets of the form $en_i(s)$ and then randomly picks one of these to return in line 8, reverting to $en(s)$ if necessary.

On the surface, random choice seems just as arbitrary as first choice. However, in Section 4 the same partial order reduction run is repeated many times with the random choice approach. This allows us to measure experimentally how sensitive the reduction is to the choice of ample set.

3.3 Using SCCs for Ample Sets

One drawback of the approach in the previous section is that the ample set contains the enabled transitions of either all or exactly one of the processes. It is easy to imagine a scenario of four processes $P_1 \dots P_4$ where en_1 and en_2 are mutually dependent, and en_3 and en_4 are mutually dependent, and all other pairings are independent. In this scenario it is possible to choose $ample = en_1 \cup en_2$ or $ample = en_3 \cup en_4$, but this is never done.

In [12] an algorithm that constructs a graph whose maximal strongly connected components (SCCs) are used as candidates for *ample* is presented.

Definition 9. *Let $M = (S, \hat{e}, \Sigma, \Delta)$ be the state space of $\mathcal{P}$ from $\hat{e}$, D be a dependency relation, R be a precedence relation, and $s \in S$ a state of M.*

- *For two processes P_i and P_j, if one or both of the conditions in Proposition 1 are violated, then P_j is a* conflicting process *for P_i in state s.*

```
AMPLE2(S)
1  E ← ∅
2  for i ∈ {1, ..., k} do
3      A ← true
4      for j ≠ i do
5          if pre(dis_i(s)) ∩ Σ_j ≠ ∅ or dep(en_i(s)) ∩ Σ_j ≠ ∅ then
6              A ← false
7              E ← E ∪ {(i, j)}
8      if A ∧ en_i(s) ≠ ∅ then return en_i(s)
9  return en_H(s) where H is some SCC of G_s = ({1, ..., k}, E)
       that satisfies the conditions of Proposition 2
```

Fig. 2. Ample set selection using a conflict graph

- *$G_s = (W, E)$ is a* conflict graph *for state s such that the vertices are process indices: $W = \{1, \ldots, k\}$ and $(i, j) \in E$ if and only if P_j is a conflicting process for P_i in state s.*
- *If H is an SCC of the conflict graph G_s, then $en_H(s) = \cup_{i \in H} en_i(s)$.* □

Then we have the following:

Proposition 2. *Let $M = (S, \hat{e}, \Sigma, \Delta)$ be the state space of $\mathcal{P}$ from $\hat{e}$, $s \in S$ be some state of M, and G_s be the conflict graph for state s. If H is an SCC of G_s such that*

1. *$en_H(s) \neq \emptyset$, and*
2. *for all SCCs $H' \neq H$ that are reachable from H, $en_{H'}(s) = \emptyset$,*

then $en_H(s)$ is an ample set for state s. □

This gives us the correctness of the algorithm in Figure 2.

To see how this approach can improve upon AMPLE1, consider the model of the philosophers' banquet, shown in Figure 3. The whole system consists of two completely independent copies of the classic four dining philosophers system, as illustrated in Figure 3(b). (The details of a single philosopher are shown in Figure 3(a).)

No reduction is possible with AMPLE1, because each $en_i(s)$ set of each philosopher contains transitions that are dependent on the transitions of the philosopher to the left or right, and is therefore invalid according to Proposition 1. On the other hand, AMPLE2 is able to select an ample set $\cup_{i \in \text{Table1}} en_i(s)$. The full system has 6400 states and 33920 transitions, which AMPLE2 reduces to 95 states and 152 transitions; as mentioned, AMPLE1 does not reduce the state space at all.

As in the case of AMPLE1, we shall consider three versions of AMPLE2:

- *First choice:* The algorithm as it stands.
- *Minimum choice:* The algorithm modified to compute all valid en_H, and to return the smallest such candidate.
- *Random choice:* The algorithm modified to compute all valid en_H, and to return a random candidate.

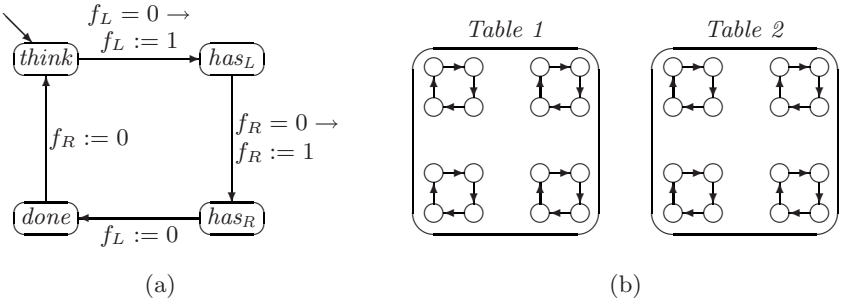

Fig. 3. Model of the philosophers' banquet. (a) Details of a single philosopher; f_L and f_R refer to a philosopher's left and right forks. (b) A banquet consisting of two independent copies of the classic dining philosophers model.

4 Experimental Results

The previous section presented three different versions of the D and R relations, two different algorithms (AMPLE1 and AMPLE2) that use the relations to compute potential ample sets, and three different ways (*first*, *minimum*, and *random*) of choosing an actual ample set.

Each of the 18 combinations of techniques are evaluated by applying them to models taken from the BEEM repository [8]. The full repository contains 300 variants of 57 basic models, and covers a range of genres: protocols, mutual exclusion and leader election algorithms, hardware control, scheduling and planning, and others. Since the experiments are long-running, only the 114 smallest models were chosen, but with at least one variant for each basic model. The sizes of the models range from 80 to 124 704 states, and 212 to 399 138 transitions.

For each model, the original model source code is converted to a C program that generates the full state space and writes the resulting graph (without detailed state contents) to a file. The D_s/R_s and D_f/R_f relations are also computed and stored along with the graphs. In all cases the full state space is identical to those described on the BEEM website.

All the numbers in the tables that follow refer to the percentage of states in the full state space explored by a method. This provides some measure of savings with respect to memory consumption, independent of specific data structures and architecture. The overhead costs involved in calculating ample sets make it hard to give a similarly independent measure for computation time.

4.1 Conflict Graph in the Static Case

The first question we address is whether the use of the conflict graph makes any difference. Algorithm AMPLE2 never fares worse than algorithm AMPLE1; Table 1(a) shows those cases where it fares better. For all but a small number of models its impact is negligible.

Table 1. Comparison of AMPLE1 and AMPLE2 algorithms in the static case. (a) Instances where AMPLE1 and AMPLE2 differ. (b) Further instances where AMPLE1 or AMPLE2 or both achieve reduction.

(a)

Model	AMPLE1 D_s/R_s	AMPLE2 D_s/R_s
`phils3`	100.00	32.37
`trgate2`	100.00	54.77
`trgate3`	100.00	55.61
`trgate1`	100.00	59.12
`phils1`	100.00	60.00
`bopdp1`	100.00	95.74
`elev21`	100.00	96.30

(b)

Model	AMPLE1 D_s/R_s	AMPLE2 D_s/R_s
`mcs4`	5.88	5.88
`anders4`	46.47	46.47
`anders2`	61.55	61.55
`peters1`	62.25	62.25
`mcs2`	65.06	65.06
`szyman1`	69.46	69.46
`ldrfil2`	76.57	76.57
`peters2`	82.42	82.42
`ldrfil4`	82.71	82.71
`mcs1`	89.30	89.30
`ldrfil3`	96.39	96.39
`krebs2`	97.60	97.60

This is partly due to the fact that the D_s/R_s relations are crude overapproximations of the true dependency between transitions, that do not provide much information for either AMPLE1 or AMPLE2 to exploit. Table 1(b) shows all further models for which any reduction at all was achieved. For the remaining 92 models both AMPLE1 and AMPLE2 explored the entire state space.

In all tables the rows are ordered according to the reduction reported in the rightmost column. In some tables we omit results for the same model with a different parameterization, due to lack of space.

4.2 The Static v. Full Calculation of D/R

With the exception of the `msc4` model, the results in Tables 1(a) and (b) could be seen as disappointing. On the other hand, they demonstrate what can be achieved with very basic static analysis.

The question that arises is how much can be gained by refining the D/R relations. There are essentially two ways of achieving this.

Firstly, a more sophisticated analysis of the structural transition guards and assignments can eliminate unnecessary dependencies between some transitions. Such an analysis may include, for example, the kind of reasoning employed to conclude that transitions t_a and t_b in Section 3.1 are independent of one another. This is what Godefroid and Pirottin call *refined dependency* [5].

Secondly, some dependencies can be computed on-the-fly. For example, in models of concurrency with asynchronous communication, the emptiness or fullness of a bounded-length communication channel affects the dependency of some operations on the channel. Godefroid and Pirottin refer to this approach as *conditional*

Table 2. Comparison of static and full D/R relations

Model	AMPLE2 D_s/R_s	AMPLE1 D_f/R_f	AMPLE2 D_f/R_f	Model	AMPLE2 D_s/R_s	AMPLE1 D_f/R_f	AMPLE2 D_f/R_f
cycsch1	100.00	1.19	1.19	prots3	100.00	72.45	70.75
mcs4	5.88	4.13	4.13	peters2	82.42	71.54	71.54
fwtree1	100.00	6.25	6.25	drphls2	100.00	72.25	72.25
phils3	32.37	29.63	10.84	collsn2	100.00	73.94	73.94
mcs1	89.30	18.35	18.35	prodcl1	100.00	74.08	74.08
anders4	46.47	22.78	22.78	teleph1	100.00	75.16	75.16
iprot2	100.00	26.16	25.97	lamp3	100.00	75.18	75.18
mcs2	65.06	34.45	34.45	fwlink1	100.00	81.61	78.83
phils1	60.00	60.00	47.50	pgmprt4	100.00	80.82	80.82
fwlink2	100.00	51.37	50.66	ldrfil3	96.39	83.20	83.20
krebs1	100.00	90.06	51.09	bopdp2	95.35	85.81	84.78
ldrelc3	100.00	54.26	54.26	fischr1	100.00	87.38	87.38
teleph2	100.00	59.55	59.55	bakery3	100.00	87.51	87.51
ldrelc1	100.00	60.52	60.52	exit2	100.00	100.00	87.78
szyman1	69.46	62.98	62.98	brp21	100.00	87.96	87.96
prodcl2	100.00	63.12	63.12	pubsub1	100.00	94.48	88.97
at1	100.00	65.71	65.35	fwtree2	100.00	98.93	89.43
szyman2	72.22	65.76	65.76	pgmprt2	100.00	89.49	89.49
ldrfil2	76.57	65.94	65.94	brp2	100.00	99.51	96.46
lamp1	100.00	66.24	66.24	extinc2	100.00	98.95	96.87
prots2	100.00	72.51	67.69	cycsch2	100.00	100.00	98.55
collsn1	100.00	68.46	68.46	synaps2	100.00	100.00	99.75
drphls1	100.00	68.54	68.54	plc1	100.00	99.95	99.95

dependency [5], and a related technique is briefly described by Peled [9] and more fully by Holzmann [6].

Both these approaches are subsumed by D_f/R_f as defined in Section 3.1. In other words, the full versions of the D and R relations supply an upper bound to the reduction that can be achieved with refined and conditional dependency. Table 2 compares the AMPLE2 algorithm for the static D/R, and the AMPLE1 and AMPLE2 algorithms for the full D/R.

As the table shows, a significant reduction can be achieved for some models, even when using AMPLE1. Nevertheless, the SCC approach is able to exploit the improved dependency relations even further.

These results lead to a further question: given the choice of improving D or improving R, which of the two relations should be refined, if possible? To answer this question, we combined the *static* version of D with the *full* version of R, and vice versa. The results of these experiments are shown in Table 3.

Only for the four topmost models in the table does the D_s/R_f combination make a greater impact than the D_f/R_s combination. In all other cases, a more accurate version of D leads to greater reduction.

Table 3. Combinations of static/full D/R

Model	AMPLE2 D_s/R_s	AMPLE2 D_s/R_f	AMPLE2 D_f/R_s	AMPLE2 D_f/R_f
`fwtree1`	100.00	9.19	100.00	6.25
`bopdp2`	95.35	93.24	95.11	84.78
`bopdp1`	95.74	93.42	95.33	86.88
`exit2`	100.00	87.78	100.00	87.78
`phils3`	32.37	32.37	10.84	10.84
`mcs1`	89.30	89.30	20.11	18.35
`anders4`	46.47	46.47	22.78	22.78
`iprot2`	100.00	100.00	27.40	25.97
`iprot1`	100.00	100.00	30.82	28.94
`mcs2`	65.06	65.06	38.28	34.45
`brp1`	100.00	100.00	99.15	96.98
`brp2`	100.00	100.00	99.51	96.46
`plc1`	100.00	100.00	99.97	99.95

Table 4. Comparison of the full and dynamic D/R relations

Model	AMPLE2 D_f/R_f	AMPLE2 D_d/R_d
`mcs2`	34.45	22.73
`fwtree2`	89.43	28.51
`ldrflt1`	88.70	50.62
`cycsch2`	98.55	53.73
`prots3`	70.75	70.18
`pubsub1`	88.97	88.10
`needhm1`	100.00	89.54
`bakery1`	94.42	93.63
`bakery2`	100.00	98.87
`gear1`	100.00	99.40

4.3 Dynamic Version of D/R

The effect of using the dynamic version of D/R is compared to the full version in Table 4. For the majority of the 114 models, little further reduction is achieved, and only a couple of models (`mcs2`, `fwtree2`, `ldrflt1`, `cycsch2`, and `needhm1`) exhibit significant reduction (more than 10%). Of course, the use of D_f/R_f already reduces the state space for many models, and there is less "room" for additional reduction. Furthermore, if the state space of a system is strongly connected, the D_f and D_d relations are identical, as are R_f and R_d.

Table 5. Comparison of the first, minimum, and random choice

Model	AMPLE2, D_f/R_f 1st	min	random
fwtree1	6.25	6.25	6.25....6.99
mcs1	18.35	18.35	18.05...18.37
iprot2	25.97	25.98	27.48...29.61
iprot1	28.94	28.96	29.78...34.00
mcs2	34.45	34.45	34.38...34.87
fwlink2	50.66	50.68	50.66...50.68
ldrelc3	54.26	54.28	54.12...54.21
ldrelc1	60.52	60.55	60.27...60.57
szyman1	62.98	62.98	63.12...63.33
prodcl2	63.12	63.12	63.30...64.06
ldrelc2	64.38	64.05	64.07...64.20
krebs2	65.17	65.17	65.18...65.22
at1	65.35	65.35	65.35...65.38
krebs1	51.09	47.95	57.92...65.51

Model	AMPLE2, D_f/R_f 1st	min	random
ldrflt2	65.94	65.94	65.86...65.99
szyman2	65.76	65.76	66.05...66.22
drphls1	68.54	68.54	68.54...68.55
prots2	67.69	68.25	68.99...69.52
drphls2	72.25	72.25	72.30...72.36
at2	72.73	72.73	72.73...72.75
prots3	70.75	72.31	71.28...73.09
prodcl1	74.08	74.08	74.40...75.45
prots1	78.77	78.11	78.27...79.51
pgmprt4	80.82	80.82	80.87...80.92
fwlink4	82.06	82.06	82.45...82.73
collsn2	73.94	73.94	83.09...85.50
exit2	87.78	87.10	87.84...88.01
fischr1	87.38	87.38	86.75...88.17

Model	AMPLE2, D_d/R_d 1st	min	random
mcs2	22.73	22.87	23.08...24.43
fwtree2	28.51	28.76	31.54...35.35
ldrflt1	50.62	50.28	50.24...52.17
cycsch2	53.73	57.42	54.09...55.15
prots3	70.18	72.31	71.67...72.91
prots1	78.77	78.11	78.44...79.42

Model	AMPLE2, D_d/R_d 1st	min	random
teleph1	75.78	70.00	77.42...82.97
pubsub1	88.10	88.10	87.41...88.10
needhm1	89.54	89.54	89.34...91.15
bakery1	93.63	93.63	93.63...94.02
bakery2	98.87	98.87	98.95...99.48
gear1	99.40	99.55	99.44...99.70

4.4 First, Minimum, and Random Choice

Lastly, Table 5 shows the results of experiments in which a different choice of valid ample sets is exercised. In the case of D_f/R_f, the choice of first, minimum, and random ample sets produces no effect for either the AMPLE1 or AMPLE2 approaches. This may be explained by the fact that the choices are so limited that it does not matter which ample set is selected.

In the case of the full and dynamic versions of D/R, however, some variation can be observed. For six models, selecting the SCC with the smallest number of enabled transitions produces an improvement in the reduction; their names are shown in italics. Note, however, that this strategy does not consistently improve the reduction and that the same model behaved differently in the full and dynamic versions. The improvement is largest for `teleph1` (-5.78%) and for `krebs1` (-3.14%); for the other models it is less than 1%. For 12 models the minimum choice leads to losses of reduction, although these are generally smaller than the improvements.

The situation is somewhat similar when a valid SCC is chosen at random. Each experiment was repeated 50 times for D_f/R_f and 20 times for D_d/R_d to

produce the results in Table 5. In the case of D_f/R_f, this strategy produces a range of 7.59% for the `krebs1` model, and for D_d/R_d a range of 5.55% for the `teleph1` model. All other ranges are smaller than 5% for the remaining models shown, and zero for the rest.

The relatively small ranges seem to indicate that in the majority of cases, reduction is not overtly sensitive to the choice of ample set. However, this does not rule out the possibility that more advanced, systematic heuristics for choosing an ample set could produce significant savings.

5 Discussion

The use of partial order reduction is widespread, and many improvements to the basic techniques have been proposed. Before such proposals are pursued, it is worthwhile to try to determine whether any significant improvement is possible at all. This paper has attempted to partially address this question. We have

- presented empirical lower bounds for partial order reduction based on a rough approximation of the dependency relation between transitions;
- presented empirical upper bounds based on information derived from the full state space;
- demonstrated that it is possible to improve reduction using a relatively simple technique such as a conflict graph that exploits information about transition dependency more fully than the standard technique; and
- shown that, given a choice of ample sets, choosing the smallest set, or a random set does not lead to significantly greater reduction in any of our experiments.

It is important to point out that it is unlikely that the upper bounds we present here are achievable. We have left the effect of the cycle-closing conditions and visibility for future work. Both tend to reduce the effect of partial order reduction. Generally speaking, the reduction does not appear to be as significant as reported elsewhere.

References

1. Clarke, E.M., Grumberg, O., Minea, M., Peled, D.: State space reduction using partial order techniques. Software Tools for Technology Transfer 2(3), 279–287 (1999)
2. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
3. Flanagan, C., Godefroid, P.: Dynamic partial-order reduction for model checking software. In: Palsberg, J., Abadi, M. (eds.) Proceedings of the 32nd Annual ACM Symposium on Principles of Programming Languages, January 2005, pp. 110–121 (2005)
4. Godefroid, P.: Partial-order Methods for the Verification of Concurrent Systems: an Approach to the State-explosion Problem. LNCS, vol. 1032. Springer, Heidelberg (1996)

5. Godefroid, P., Pirottin, D.: Refining dependencies improves partial-order verification methods (extended abstract). In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 438–449. Springer, Heidelberg (1993)
6. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley, Reading (2004)
7. Holzmann, G.J., Peled, D.: An improvement in formal verification. In: Hogrefe, D., Leue, S. (eds.) Proceedings of the 7th IFIP TC6/WG6.1 International Conference on Formal Description Techniques (FORTE 1994), June 1994, pp. 197–211 (1994)
8. Pelánek, R.: BEEM: Benchmarks for explicit model checkers. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 263–267. Springer, Heidelberg (2007), http://anna.fi.muni.cz/models/
9. Peled, D.: Combining partial order reductions with on-the-fly model-checking. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 377–390. Springer, Heidelberg (1994)
10. Peled, D., Valmari, A., Kokkarinen, I.: Relaxed visibility enhances partial order reduction. Formal Methods in System Design 19(3), 275–289 (2001)
11. Peled, D.A.: All from one, one for all: on model checking using representatives. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 409–423. Springer, Heidelberg (1993)
12. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design 1(1), 297–322 (1992)
13. Varpaaniemi, K.: On the Stubborn Set Method in Reduced State Space Generation. PhD thesis, Digital Systems Laboratory, Helsinki University of Technology (May 1998)