Partial Order Reduction in Directed Model Checking

Alberto Lluch-Lafuente, Stefan Edelkamp, and Stefan Leue

Institut für Informatik, Albert-Ludwigs-Universität
Georges-Köhler-Allee, D-79110 Freiburg
eMail: {lafuente,edelkamp,leue}@informatik.uni-freiburg.de

Abstract. Partial order reduction is a very succesful technique for avoiding the state explosion problem that is inherent to explicit state model checking of asynchronous concurrent systems. It exploits the commutativity of concurrently executed transitions in interleaved system runs in order to reduce the size of the explored state space. Directed model checking on the other hand addresses the state explosion problem by using guided search techniques during state space exploration. As a consequence, shorter errors trails are found and less search effort is required than when using standard depth-first or breadth-first search. We analyze how to combine directed model checking with partial order reduction methods and give experimental results on how the combination of both techniques performs.

1 Introduction

Model checking [3] is a formal analysis technique for the verification of hardware and software systems. Given the model of the system as well as a property specification, typically formulated in some temporal logic formalism, the state space of the model is analyzed to check whether the property is valid or not. The main limitation of this method is the size of the resulting state space, known as the *state explosion problem*. It occurs due to non-determinism in the model introduced by data or concurrency.

Different approaches have been proposed to tackle this problem. One of the most successful techniques is partial order reduction [22]. This method explores a reduced state space by exploiting the independence of concurrently executed events. Partial order reduction is particularly efficient in asynchronous systems, where many interleavings of concurrent events are equivalent with respect to a given property specification. Considering only one or a few representatives of one class of equivalent interleavings leads to drastic reductions in the size of the state space to be explored.

Another technique that has been suggested in dealing with the state explosion problem is the use of heuristic search techniques. It applies state evaluation functions to rank the set of successor states in order to decide where to continue the search. Applying such methods often allows to find errors at optimal or sub-optimal depths and to find errors in models for which "blind" search strategies like depth-first and breadth-first search exceed the available time and space resources. Optimal or near-to optimal solutions are particularly important for designers to understand the sequence of steps that lead to an error, since shorter trails are likely to be more comprehensible than longer ones. In protocol verification, heuristic search model checking has been shown to accelerate the search for finding errors [7] and to shorten already existing long trails [6].

It is not a priori obvious to what extent partial order reduction and guided search can co-exist in model checking. In fact, as we show later, applying partial-order reduction to a state space does not preserve optimiality of the shortest path to a target state. It is the goal of this paper to show that nevertheless, partial order reduction and directed model checking can co-exist, and that the mutual negative influence is only minimal.

In this paper, we will focus on safety error detection in model checking. We will establish a hierarchy of relaxation of the cycle condition for partial order reduction known as C3, and we will classify the relaxations with respect to their applicability to different classes of heuristic search algorithms. To the best of our knowledge, at the time of writing no publication addressing heuristic search in model checking [7,8,6,12,4,17,24] has analyzed how to combine guided search with partial order reduction.

The paper is structured as follows. Section 2 gives some background on directed model checking. Section 3 discusses partial order reduction and a hierarchy of conditions for its application to different search algorithms. This Section also addresses the problem of optimality in the length of the counterexamples. Section 4 presents experimental results showing how the combination of partial order reduction and directed model checking perform. Section 5 summarizes the results and concludes the paper.

2 Directed Model Checking

Analysts have different expectations regarding the capabilities of formal analysis tools at different stages of the software process [4]. In *exploratory* mode, usually applicable to earlier stages of the process, one wishes to find errors fast. In *fault-finding* mode, which usually follows later, one expects to obtain meaningful error trails while one is willing to tolerate somewhat longer execution times. Directed model checking has been identified as an improvement of standard model checking algorithms that help in achieving the objectives of both modes.

Early approaches to directed model checking [17,24] propose the use of best-first search algorithms in order to accelerate the search for error states. Further approaches [8,7,6,4] propose the full spectrum of classical heuristic search strategies for the verification process in order to accelerate error detection and to provide optimal or near-to-optimal trails. Most of these techniques can be applied to the detection of safety properties only or for shortening given error traces corresponding to liveness violations [6].

Contrary to blind search algorithms like depth- and breadth-first search, heuristic search exploits information of the specific problem being solved in order to guide the search. Estimator functions approximate the distance from a given state to a set of goal states. The values provided by these functions decide in which direction the search will be continued. Two of the most frequently used heuristic search algorithms are A* [10] and IDA* [15]. In the following we describe a general state expanding search algorithm that can be either instantiated as a depth or breadth first search algorithm or as one of the above heuristic search algorithms. We briefly the basic principles of heuristic search algorithms, and consider different heuristic estimates to be applied in the context of directed model checking for error detection. In our setting we interpret error states as goal nodes in an underlying graph representation of the state space with error trails corresponding to solution paths.

2.1 General State Expanding Search Algorithm

The general state expanding search algorithm of Figure 1 divides the state space S into three sets: the set *Open* of visited but not yet expanded states, the set *Closed* of visited and expanded states and the set $S \setminus (Closed \cup Open)$ of not yet visited states. The algorithm performs the search by extracting states from *Open* and moving them into *Closed*. States extracted from *Open* are expanded, i.e. their successor states are generated. If a successor of an expanded state is neither in *Open* nor in *Closed* it is added to *Open*.

```
procedure search
  Closed ← ∅
  Open ← ∅
  Open.insert(start_state)
  while (Open ≠ ∅) do
    u ← Open.extract()
    Closed.insert(u)
    if error(u) then
      exit ErrorFound
    for each successor v of u do
      if not v ∈ Closed ∪ Open then
          Open.insert(v)
```

Fig. 1. A general state expanding search algorithm.

Breadth-first search and depth-first search can be defined as concrete cases of the general algorithm presented above, where the former implements *Open* as a FIFO queue and the latter as a stack.

2.2 Algorithm A*

Algorithm A* treats *Open* as a priority queue in which the priority of a state v is given by function $f(v)$ that is computed as the sum of the length of the path from the start state $g(v)$ and the estimated distance to a goal state $h(v)$. In addition to the general algorithm, A* can move states from *Closed* to *Open* when they are reached along a shorter path. This step is called reopening and is necessary to guarantee that the algorithm will find the shortest path to the goal state when non-monotone heuristics are used. For the sake of simplicity, throughout the paper we only consider monotone heuristics which do not require reopening [20]. Monotone heuristics satisfy that for each state u and each successor v of u the difference between $h(u)$ and $h(v)$ is less or equal to the cost of the transition that goes from u to v. Assuming monotone estimates is not a severe restriction, since it turns out that in practical examples most proposed heuristics are indeed monotone. If h is a lower bound of the distance to a goal state, then A* is admissible, which means that it will always return the shortest path to a goal state [19]. Best-first search is a common variant of A* that takes only h into account.

2.3 Iterative-deepening A*

*Iterative-deepening A**, IDA* for short, is a refinement of the brute-force depth-first iterative deepening search (DFID) that combines the space efficiency of depth-first search and the admissibility of A*. While DFID performs successive depth-first search iterations with increasing depth bound, in IDA* increasing cost bounds are used to limit search iterations. The cost bound f of a state is the same as in A*. Similar to A*, IDA* guarantees optimal solution paths if the estimator is a lower bound.

2.4 Heuristic Estimates

The above presented search algorithms require suitable estimator functions. In model checking, such functions approximate the number of transitions for the system to reach an error state from a given state. During the model checking process, however, an explicit error state is not always available. In fact, in many cases we do not know if there is an error in the model at all. We distinguish the cases when errors are unknown and when error states are explicit.

If an explicit error state is given, estimates that exploit the information of this state can be devised. Examples are estimates based on the Hamming distance of the state vectors for the current and the targe state, and the FSM distance that uses the minimum local distances in the state transition graph of the different processes to derive an estimat [6].

Estimating the distance to *unknown* error states is more difficult. The formula-based heuristic [7] constructs a function that characterizes error states. Given an error formula f and starting from state s, a heuristic function $h_f(s)$ is constructed for estimating the number of transitions needed until a state s'

is reached, where $f(s')$ holds. Constructing an error formula for deadlocks is not trivial. In [7] we discuss various alternatives, including formula based approaches, an estimate based on the number of non-blocked processes, and an estimate derived from user-provided characterizations of local control states as deadlock-prone.

3 Partial Order Reduction

Partial order reduction methods exploit the commutativity of asynchronous systems in order to reduce the size of the state space. The resulting state space is constructed in such a manner that it is equivalent to the original one with respect to the specification. Several partial order approaches have been proposed, namely those based on "stubborn" sets [23], "persistent" sets [9] and "ample" sets [21]. Although they differ in detail, they are based on similar ideas. Due to its popularity, in this paper we mainly follow the ample set approach. Nonetheless, most of the reasoning presented in this paper can be adjusted to any of the other approaches.

3.1 Stuttering Equivalence of Labeled Transition Systems

Our approach is mainly focused to the verification of asynchronous systems where the global system is constructed as the asynchronous product of a set of local component processes following the interleaving model of execution. Such systems can be modeled by labeled transitions systems.

A labeled finite transition system is a tuple $\langle S, S_0, T, AP, L\rangle$ where S is a finite set of states, S_0 is the set of initial states, T is a finite set of transitions such that each transition $\alpha \in T$ is a partial function $\alpha : S \to S$, AP is a finite set of propositions and L is a labeling function $S \to 2^{AP}$. The execution of a transition system is defined as a sequence of states interleaved by transitions, i.e. a sequence $s_0\alpha_0 s_1 \ldots$, such that s_0 is in S_0 and for each $i \geq 0$, $s_{i+1} = \alpha_i(s_i)$.

The algorithm for generating a reduced state space explores only some of the successors of a state. A transition α is *enabled* in a state s if $\alpha(s)$ is defined. The set of enabled transitions from a state s is usually called the *enabled set* and denoted as $enabled(s)$. The algorithm selects and follows only a subset of this set called the *ample set* and denoted as $ample(s)$. A state s is said to be *fully expanded* when $ample(s) = enabled(s)$.

Partial order reduction techniques are based on the observation that the order in which some transitions are executed is not relevant. This leads to the concept of independence between transitions. Two transitions $\alpha, \beta \in T$ are independent if for each state $s \in S$ in which both transitions are defined the following two properties hold:

1. α and β do not disable each other: $\alpha \in enabled(\beta(s))$ and $\beta \in enabled(\alpha(s))$.
2. α and β are *commutative*: $\alpha(\beta(s)) = \beta(\alpha(s))$.

Two transitions are dependent if they are not independent. A further fundamental concept is the fact that some transitions are *invisible* with respect to atomic propositions that occur in the property specification. A transition α is invisible with respect to a set of propositions P if for each state s and for each successor s' of s we have $L(s) \cap P \ = \ L(s') \cap P$.

We now present the concept of *stuttering equivalence* with respect to a property P. A *block* is defined as a finite execution containing invisible transitions only. Intuitively, two executions are stuttering equivalent if they can be defined as a concatenation of blocks such that the atomic propositions of the i-th block of boths executions have the same intersection with P, for each $i > 0$. Figure 2 depicts two stuttering equivalent paths with respect to a property in which only propositions p and q occurr.

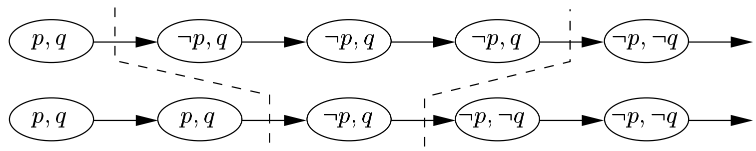

Fig. 2. Stuttering equivalent executions.

Two transition systems are stuttering equivalent if and only if they have the same set of initial states and for each execution in one of the systems starting from an initial state there exists a stuttering equivalent execution in the other system starting from the same initial state. It can be shown that LTL$_{-X}$ formulae[1] cannot distinguish between stuttering equivalent transition systems [3]. In other words, if M and N are two stuttering equivalent transition systems, then M satisfies a given LTL$_{-X}$ specification if and only if N also does.

3.2 Ample Set Construction for Safety LTL$_{-X}$

The main goal of the ample set construction is to select a subset of the successors of every state such that the reduced state space is stuttering equivalent to the full state space with respect to a property specification given by a set P of atomic propositions. The construction should offer significant reduction without requiring a high computational overhead. The following four conditions are necessary and sufficient for the proper construction of a partial order reduced state space for a given property specification P.

Condition C0: $ample(s)$ is empty exactly when $enabled(s)$ is empty.

Condition C1: Along every path in the full state space that starts at s, a transition that is dependent on a transition in $ample(s)$ does not occur without a transition in $ample(s)$ occurring first.

[1] LTL$_{-X}$ is the linear time temporal logic without the next-time operator X.

Condition C2: If a state s is not fully expanded, then each transition α in the ample set must be invisible with regard to P.

Condition C3: If for each state of a cycle in the reduced state space, a transition α is enabled, then α must be in the ample set of some of the states of the reduced state space.

Observe that the approximations used in **C0**, **C1** and **C2** can be implemented independently from the particular search algorithm used. It was shown in [3] that the complexity of checking **C0** and **C2** does not depend on the search algorithm. Checking Condition **C1** is more complicated. In fact, it has been shown to be at least as hard as checking reachability for the full state space. It is, however, usually over-approximated by checking a stronger condition [3] that can be checked independently of the search algorithm.

Condition **C3** has been implicitly proposed in [11]. In the following we focus on this condition. We will see that the complexity of checking it depends on the search algorithm used.

3.3 Dynamically Checking C3

Checking **C3** can be reduced to detecting cycles during the search. Cycles can easily be established in depth-first search: Every cycle contains a *backward edge*, i.e. an edge that links back to a state that is stored on the search stack [3]. Consequently, avoiding ample sets containing only backward edges except when the state is fully expanded ensures satisfaction of **C3** when using depth-first search or IDA*, since both methods perform a depth-first traversal. The resulting stack-based characterization $\mathbf{C3}_{stack}$ can be stated as follows[11]:

Condition $\mathbf{C3}_{stack}$: If a state s is not fully expanded, then at least one transition in $ample(s)$ does not lead to a state on the search stack.

The implementation of $\mathbf{C3}_{stack}$ for depth-first search strategies marks each expanded state on the stack with an additional flag, so that stack containment can be checked in constant time. Depth-first strategies that record visited states will not consider every cycle in the state space on the search stack, since there might exist exponentially many of them. However, $\mathbf{C3}_{stack}$ is still a sufficient condition for **C3** since every cycle contains at least a backward edge.

Detecting cycles with general state expanding search algorithms that do not perform a depth-first traversal of the state space is more complex. For a cycle to exist, it is necessary to reach an already visited state. If during the search a state is found to be already visited, checking that this state is part of a cycle requires to check if this state is reachable from itself which increases the time complexity of the algorithm from linear to quadratic in the size of the state space. Therefore the commonly adopted approach assumes that a cycle exists whenever an already visited state is found. Using this idea leads to weaker reductions, since it is known that the state spaces of concurrent systems usually have a high density of duplicate states. The resulting condition [2,1] is defined as:

Condition $\mathbf{C3}_{duplicate}$: If a state s is not fully expanded, then at least one transition in $ample(s)$ does not lead to an already visited state.

A proof of sufficiency of condition $\mathbf{C3}_{stack}$ for depth-first search is given in [11]. The proof of sufficiency of condition $\mathbf{C3}_{duplicate}$ when combined with a depth-first search is given by the fact that $\mathbf{C3}_{duplicate}$ implies $\mathbf{C3}_{stack}$; if at least one transition in $ample(s)$ has a non-visited successor this transition certainly does not lead to a successor on the stack.

In order to prove the correctness of partial order reduction with condition $\mathbf{C3}_{duplicate}$ for general state expanding algorithms in the following lemma, we will use induction on the state expansion ordering, starting from a completed exploration and moving backwards with respect to the traversal algorithm. As a by-product the more general setting in the lemma also proves the correctness of partial order reduction according to condition $\mathbf{C3}_{duplicate}$ for depth-first, breadth-first, best-first, and A* like search schemes. The lemma fixes a state $s \in S$ after termination of the search and ensures that each enabled transition is executed either in the ample set or in a state that appears later on in the expansion process. Therefore, no transition is omitted. Applying the lemma to all states s in S implies **C3**, which, in turn, ensures a correct reduction.

Lemma 1. *For each state $s \in S$ the following is true: when the search of a general search algorithm terminates, each transition $\alpha \in enabled(s)$ has been selected either in $ample(s)$ or in a state s' such that s' has been expanded after s.*

Proof. Let s be the last expanded state. Every transition $\alpha \in enabled(s)$ leads to an already expanded state, otherwise the search would have been continued. Condition $\mathbf{C3}_{duplicate}$ enforces therefore that state s is fully expanded and the lemma trivially holds for s.

Now suppose that the lemma is valid for those states whose expansion order is greater than n. Let s be the n-th expanded state. If s is fully expanded, the lemma trivially holds for s. Otherwise we have that $ample(s) \subset enabled(s)$. Transitions in $ample(s)$ are selected in s. Since $ample(s)$ is accepted by condition $\mathbf{C3}_{duplicate}$ there is a transition $\alpha \in ample(s)$ such that $\alpha(s)$ leads to a state that has not been previously visited nor expanded. Evidently the expansion order of $\alpha(s)$ is higher than n. Condition **C1** implies that the transitions in $ample(s)$ are all independent from those in $enabled(s) \setminus ample(s)$ [3]. A transition $\gamma \in enabled(s) \setminus ample(s)$ cannot be dependent from a transition in $ample(s)$, since otherwise in the full graph there would be a path starting with γ and a transition depending on some transition in $ample(s)$ would be executed before a transition in $ample(s)$. Hence, transitions in $enabled(s) \setminus ample(s)$ are still enabled in $\alpha(s)$ and contained in $enabled(\alpha(s))$. By the induction hypothesis the lemma holds for $\alpha(s)$ and, therefore, transitions in $enabled(s) \setminus ample(s)$ are selected in $\alpha(s)$ or in a state that is expanded after it. Hence the lemma also holds for s. $\square$

3.4 Statically Checking C3

In contrast to the previous approaches the method to be presented in this Section explicitly exploits the structure of the underlying interleaving system. Recall that the global system is constructed as the asynchronous composition of several components. The authors of [16] present what they call a *static* partial order reduction method based on the following observation. Any cycle in the global state space is composed of a local cycle, which may be of length zero, in the state transition graph of each component process. Breaking every local cycle breaks every global cycle. The structure of the processes of the system is analyzed before the global state space generation begins. The method is independent from the search algorithm to be used during the verification.

Some transitions are marked are with a special flag, called *sticky*. Sticky transitions enforce full expansion of a state. Marking at least one transition in each local cycle as *sticky* guarantees that at least one state in each global cycle is fully expanded, which satisfies condition **C3**. The resulting $\mathbf{C3}_{static}$ condition is defined as follows:

Condition $\mathbf{C3}_{static}$: If a state s is not fully expanded then no transition in $ample(s)$ is sticky.

Selecting one sticky transition for each local cycle is a naive approach that can be made more effective. The impact of local cycles on the set of variables of the system can be analyzed in order to establish certain dependencies between local cycles. For example, if a local cycle C_1 has an overall incrementing effect on a variable v, for a global cycle to exist, it is necessary (but not sufficient) to execute C_1 in combination with a local cycle C_2 that has an overall decrementing effect on v. In this case one can select only a sticky transition for this pair of local cycles.

3.5 Hierarchy of C3 Conditions

Figure 3 depicts a diagram with all the presented **C3** conditions. Arrows indicate which condition enforeces which other. In the rest of the paper we will say that a condition A is stronger than a condition B if A enforces B. The dashed arrow arrow from $C3_{duplicate}$ to $\mathbf{C3}_{stack}$ denotes that when search is done with a depth-first search based algorithm $\mathbf{C3}_{stack}$ enforces $C3_{duplicate}$ but not viceversa. The dashed regions contain the conditions that can be correctly used with general state expanding algorithms, and those that work only for depth-first search like algorithms. For a given algorithm, the arrows also denote that a condition will produce better or equal reduction.

3.6 Solution Quality and Partial Order

One of the goals of directed model checking is to find shortest paths to errors. Although from a practical point of view near-to optimal solutions may be sufficient

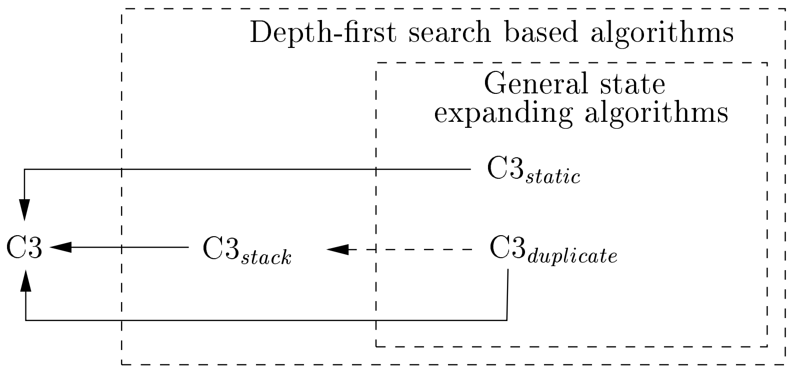

Fig. 3. C3 conditions.

to help designers during the debugging phase, finding optimal counterexamples still remains an important theoretical question. Heuristic search algorithms require lower bound estimates for guaranteeing optimal solution lengths.

Partial order reduction, however, does not preserve optimality of the solution length for the full state space. In fact, the shortest path to an error in the reduced state space may be longer than the shortest path to an error state in the full state space. Intuitively, the reason is that the concept of stuttering equivalence does not make assumptions about the length of the equivalent blocks. Suppose that the transitions α and β of the state space depicted in Figure 4 are independent and that α is invisible with respect to the set of propositions p. Suppose further that p is the only atomic proposition occurring in the safety property we want to check. With these assumptions the reduced state space for the example is stuttering equivalent to the full one. The shortest path that violates the invariant in the reduced state space is $\alpha\beta$, which has a length of 2. In the full one the path β is the shortest path to an error state and the error trail has a length of 1. Section 4 presents experimental evidence for a reduction in solution quality when applying partial order reduction.

4 Experiments

The experimental results that we report in this Section have been obtained using our experimental directed model checker HSF-SPIN[2] [7] for which we have implemented the described reduction methods. All results were produced on a SUN workstation, UltraSPARC-II CPU with 248 Mhz.

We use a set of Promela models as benchmarks including a model of a leader election protocol[3] [5] (leader), the CORBA GIOP protocol [13] (giop), a tele-

[2] Available at `www.informatik.uni-freiburg.de/~lafuente/hsf-spin`

[3] Available at `netlib.bell-labs.com/netlib/spin`

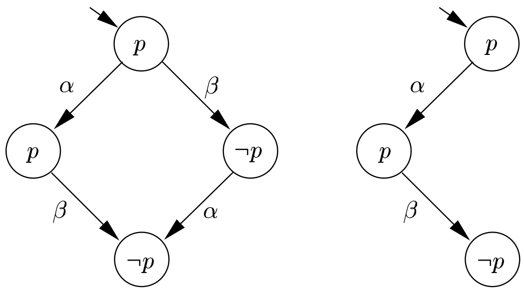

Fig. 4. Example of a full state space (left) and a reduction (right).

phony model[4] [14] (pots), and a model of a concurrent program that solves the stable marriage problem [18] (marriers). The considered versions of these protocols violate certain safety properties.

4.1 Exhaustive Exploration

The objective of the first set of experiments is to show how the different variants of the **C3** condition perform. We expect that stronger **C3** conditions according to hierarchy in Figure 3 lead to weaker reductions in the number of stored and expanded states and transitions.

Model	Reduction	States	Transitions	Time
marriers	No Reduction	96,295	264,053	20.6
	$\mathbf{C3}_{stack}$	29,501	37,341	5.5
	$\mathbf{C3}_{duplicate}$	72,536	111,170	17.5
	$\mathbf{C3}_{static}$	57,067	88,119	10.7
leader	No Reduction	54,216	210,548	36.3
	$\mathbf{C3}_{stack}$	963	4,939	4.4
	$\mathbf{C3}_{duplicate}$	1,417	6,899	5.0
	$\mathbf{C3}_{static}$	2,985	7,527	4.8
giop	No Reduction	664,376	2,579,722	259.3
	$\mathbf{C3}_{stack}$	65,964	90,870	23.1
	$\mathbf{C3}_{duplicate}$	284,083	605,147	115.0
	$\mathbf{C3}_{static}$	231,102	445,672	79.0

Table 1. Exhaustive exploration with depth-first search and several reduction methods.

We use the marriers, leader and giop protocols in our experiments. The pots model is too large to be explored exhaustively. In this and all following experi-

[4] Available at `www.informatik.uni-freiburg.de/~lafuente/models/models.html`

Model	Reduction	States	Transitions	Time	Length
marriers	no	5,077	12,455	0.93	50
	$\mathbf{C3}_{duplicate}$	2,988	4,277	0.51	50
	$\mathbf{C3}_{static}$	1,604	1,860	0.31	50
pots	no	2,668	6,519	1.57	67
	$\mathbf{C3}_{duplicate}$	1,662	3,451	1.08	67
	$\mathbf{C3}_{static}$	1,662	3,451	1.00	67
leader	no	7,172	22,876	6.87	58
	$\mathbf{C3}_{duplicate}$	65	3,190	4.76	77
	$\mathbf{C3}_{static}$	399	3,593	4.88	66
giop	no	31,066	108,971	26.50	58
	$\mathbf{C3}_{duplicate}$	21,111	48,870	16.68	58
	$\mathbf{C3}_{static}$	12,361	24,493	9.36	58

Table 2. Finding a safety violation with A* and several reduction methods.

ments we have selected the biggest configuration of these protocols that can still be exhaustively analyzed. Exploration is performed by depth-first search.

Table 1 depicts the size of the state space as a result of the application of different **C3** conditions. The number of transitions performed and the running time in seconds are also included. For each model, the first row indicates the size of the explored state space when no reduction is used.

As expected stronger conditions offer weaker reductions. This loss of reduction is especially evident in the giop protocol, where the two conditions potentially applicable in A*, namely $\mathbf{C3}_{duplicate}$ and $\mathbf{C3}_{static}$, are worse by about a factor of 4 with respect to the condition that offers the best reduction, namely $\mathbf{C3}_{stack}$.

For the marriers and giop protocols the static reduction yields a stronger reduction than condition $\mathbf{C3}_{duplicate}$. Only for the leader election algorithm this is not true. This is probably due to the relative high number of local cycles in the state transition graph of the processes in this model, and to the fact that there is no global cycle in the global state space. Since our implementation of the static reduction considers only the simplest approach where one transition in each cycle is marked as sticky, we assume that the results will be even better with refined methods for characterizing transitions as sticky.

In addition to the reduction in space consumption, partial order reduction also provides reduction in time. Even though the overhead introduced by the computation of the ample set and the static computation prior to the exploration when $\mathbf{C3}_{static}$ is used, time reduction is still achieved in all cases.

4.2 Error Finding with A* and Partial Order Reduction

The next set of experiments is intended to highlight the impact of various reduction methods when detecting errors with A*. More precisely, we want to compare

Model	Reduction	States	Transitions	Time	Length
marriers	no	4,724	84,594	19.29	50
	yes	1,298	4,924	8.40	50
pots	no	2,422	46,929	36.52	67
	yes	1,518	20,406	28.37	67
leader	no	6,989	141,668	210.67	56
	yes	55	50,403	73.90	77
giop	no	30,157	868,184	225.54	58
	yes	7,441	102,079	78.43	58

Table 3. Finding a safety violation with IDA* with and without reduction.

the two **C3** conditions $\mathbf{C3}_{duplicate}$ and $\mathbf{C3}_{static}$ that can be applied jointly with A*.

Table 2 shows the effect of applying $\mathbf{C3}_{duplicate}$ and $\mathbf{C3}_{static}$ in conjunction with A*. The table has the same format as the previous one, but this time the length of the error trail is included. Similar to SPIN[5], we count a sequence of atomic steps (respectively expansions) as one unique transition (expansion), but length of the error trail is given in steps in order to provide a better idea of what the user of the model checker gets.

As expected, both conditions achieve a reduction in the number of stored states and transitions performed. Solution quality is only lost in the case of leader. This occurs also in experiments done with IDA*. In the same test case $\mathbf{C3}_{static}$ requires the storage of more states and the execution of more transitions than $\mathbf{C3}_{duplicate}$. The reasons are the same as the ones mentioned in the previous set of experiments. On the other hand, $\mathbf{C3}_{duplicate}$ produces a longer error trail. A possible interpretation is that more reduction leads to higher probability that the anomaly that causes the loss of solution quality occurs. In other words, the bigger the reduction is, the longer the stuttering equivalent executions and, therefore, the longer the expected trail lengths become. Table 2 also shows that the overhead introduced by partial order reduction and heuristic search does not avoid time reduction.

4.3 Error Finding with IDA* and Partial Order Reduction

We also investigated the effect of partial order reduction when the state space exploration is performed with IDA*. The test cases are the same of the previous Section. Partial order reduction with IDA* uses the cycle condition $\mathbf{C3}_{stack}$.

Table 3 depicts the results on detecting a safety error with and without applying partial order reduction. The table shows the total number of transitions performed, the maximal peak of stored states and the length of the provided counterexamples. As in the previous set of experiments, solution quality is only

[5] Available at `netlib.bell-labs.com/netlib/spin/whatispin.html`

lost when applying partial order reduction in the leader election algorithm. On the other hand, this is also the protocol for which the best reduction is obtained. We assume that the reason is the same as indicated in the previous set of experiments. In addition, the overhead introduced by partial order reduction and heuristic search does avoid time reduction as explained for previous experiments.

4.4 Combined Effect of Heuristic Search and Partial Order

In this Section we are interested in analyzing the combined state space reduction effect of partial order reduction and heuristic search. More precisely, we have measured the reduction ratio (size of full state space vs. size of reduced state space) provided by one of the techniques when the other technique is enabled or not, as well as the reduction ratio of using both techniques simultaneously.

marriers	N	C
H	2.3	6.5
PO	40.8	117.6
H+PO	267.0	

pots	N	C
H	5.9	8.4
PO	1.4	1.6
H+PO	9.5	

leader	N	C
H	1.9	2.6
PO	2.7	3.2
H+PO	5.9	

giop	N	C
H	1.3	1.3
PO	2.6	2.5
H+PO	3.3	

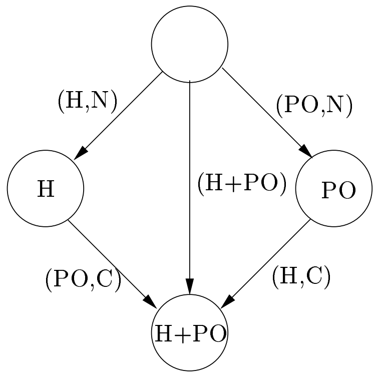

Fig. 5. Table with reduction factor due partial order and heuristic search (left) and an explanatory diagram (right).

The Table on the left of Figure 5 indicates the reduction factor achieved by partial order and heuristic search when A* is used as the search algorithm. The Figure also includes a diagram that helps to understand the table. The reduction factor due to a given technique is computed as the number of stored states when the search is done without applying the respective technique divided by the number of stored states when the search is done applying the technique. Recall that when no heuristic is applied, A* performs breadth-first search. A search is represented in the diagram by a circle labeled with the applied technique(s), namely heuristic search (H), partial-order reduction (PO) or both (H+PO). The labels of the edges in the diagram refer to the cells of the table which contain the measured reduction factors. The leftmost column of the table indicates the technique(s) for which the reduction effect is measured. When testing the reduction ratios of the methods separately, we distinguish whether the other method is applied (C) or not (N).

In some cases the reduction factor provided by one of the techniques when working alone ((H,N) and (PO,N)) improves when the other technique is applied ((H,C) and (PO,C)). This is particularly evident in the case of the marriers model, where the reduction provided by heuristic search is improved from 2.3 to 6.5 and that of partial order reduction increases from 40.8 to 117.6. The expected gain of applying both independently would be $2.3 \times 40.8 = 93.8$ while the combined effect is a reduction of 267.0 which indicates a synergetic effect. However, as illustrated by the figures for the giop model, synergetic gains cannot always be expected.

5 Conclusions

When combining partial order reduction with directed search two main problems must be considered. First, common partial order reduction techniques require to check a condition which entails the detection of cycles during the construction of the reduced state space. Depth-first search based algorithms like IDA* can easily detect cycles during the exploration. On the other side, heuristic search algorithms like A* are not well-suited for cycle detection. Stronger cycle conditions or static reduction methods have to be used. We have established a hierarchy of approximation conditions for ample set condition **C3** and our experiments show that weaker the condition, the better the effect on the state space search.

Second, partial order reduction techniques do not preserve optimality of the length of the path to error states. In other words, when partial order is used there is no guarantee to find the shortest counterexample that lead to an error, which is one of the core objectives of the paradigm of directed model checking. In current work we are analyze the possibility of avoiding this problem by exploiting independence of events to shorten error trails.

Experimental results show that in some instances, partial order reduction has positive effects when used in combination with directed search strategies. Although solution quality is lost in some cases, significant reductions can be achieved even when using A* with weaker methods than classical cycle conditions. Static reduction, in particular, seems to be more promising than other methods applicable with A*. Partial order reduction provides drastic reductions when error detection is performed by IDA*. We have also analyzed the combined effect of heuristics and reduction, showing than in most cases the reduction effect of one technique is lightly accentuated by the other. Experimental results also show that both techniques reduce running time even though the overhead they introduce.

Acknowlegdements

The authors wish to thank the reviewers for their careful reviews and for their constructive criticism. The first two authors acknowledge support they received from the Deutsche Forschungsgemeinschaft through grants Ot 64/13-2 and Ed 74/2-1.

References

1. R. Alur, R. Brayton, T. Henzinger, S. Qaderer, and S. Rajamani. Partial-order reduction in symbolic state space exploration. In *Computer Aided Verification (CAV)*, Lecture Notes in Computer Science, pages 340–351. Springer, 1997.
2. C.-T. Chou and D. Peled. Formal verification of a partial-order reduction technique for model checking. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 241–257, 1996.
3. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
4. J. M. Cobleigh, L. A. Clarke, and L. J. Osterweil. The right algorithm at the right time: Comparing data flow analysis algorithms for finite state verification. In *Proceedings of the 23^{rd} ICSE*, pages 37–46, 2001.
5. D. Dolev, M. Klawe, and M. Rodeh. An O(n log n) unidirectional distributed algorithm for extrema finding in a circle. *Journal of Algorithms*, 1982.
6. S. Edelkamp, A. L. Lafuente, and S. Leue. Trail-directed model checking. In S. D. Stoller and W. Visser, editors, *Electronic Notes in Theoretical Computer Science*, volume 55. Elsevier Science Publishers, 2001.
7. S. Edelkamp, A. Lluch-Lafuente, and S. Leue. Directed model-checking in HSF-SPIN. In *8th International SPIN Workshop on Model Checking Software*, Lecture Notes in Computer Science 2057, pages 57–79. Springer, 2001.
8. S. Edelkamp, A. Lluch-Lafuente, and S. Leue. Protocol verification with heuristic search. In *AAAI-Spring Symposium on Model-based Validation of Intelligence*, pages 75–83, 2001.
9. P. Godefroid. Using partial orders to improve automatic verification methods. In E. M. Clarke, editor, *Proceedings of the 2nd International Conference on Computer-Aided Verification (CAV '90), Rutgers, New Jersey, 1990*, number 531, pages 176–185, Berlin-Heidelberg-New York, 1991. Springer.
10. P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for heuristic determination of minimum path cost. *IEEE Transactions on on Systems Science and Cybernetics*, 4:100–107, 1968.
11. G. Holzmann, P. Godefroid, and D. Pirottin. Coverage preserving reduction strategies for reachability analysis. In *Proc. 12th Int. Conf on Protocol Specification, Testing, and Verification, INWG/IFIP*, Orlando, Fl., June 1992.
12. G. J. Holzmann. Algorithms for automated protocol verification. *AT&T Technical Journal*, 69(2):32–44, Feb. 1990. Special Issue on Protocol Testing, Specification, and Verification.
13. M. Kamel and S. Leue. Formalization and validation of the General Inter-ORB Protocol (GIOP) using PROMELA and SPIN. *International Journal on Software Tools for Technology Transfer*, 2(4):394–409, 2000.
14. M. Kamel and S. Leue. VIP: A visual editor and compiler for v-Promela. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science, pages 471–486. Springer, 2000.
15. R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.
16. R. P. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigun. Static partial order reduction. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 345–357, 1998.

17. F. J. Lin, P. M. Chu, and M. Liu. Protocol verification using reachability analysis: the state space explosion problem and relief strategies. *ACM*, pages 126–135, 1988.
18. D. McVitie and L. Wilson. The stable marriage problem. *Communications of the ACM*, 14(7):486–492, 1971.
19. N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Co., Palo Alto, California, 1980.
20. J. Pearl. *Heuristics*. Addison-Wesley, 1985.
21. D. A. Peled. Combining partial order reductions with on-the-fly model-checking. *Formal Methods in Systems Design*, 8:39–64, 1996.
22. D. A. Peled. Ten years of partial order reduction. In *Computer Aided Verification*, number 1427 in Lecture Notes in Computer Science, pages 17–28. Springer, 1998.
23. A. Valmari. A stubborn attack on state explosion. *Lecture Notes in Computer Science*, 531:156–165, 1991.
24. C. H. Yang and D. L. Dill. Validation with guided search of the state space. In *Conference on Design Automation (DAC)*, pages 599–604, 1998.