A General Approach to Partial Order Reductions in Symbolic Verification (Extended Abstract)

Parosh Aziz Abdulla[1], Bengt Jonsson[1], Mats Kindahl[1,2], and Doron Peled[3]

[1] Uppsala University, Dept. of Computer Systems,
P.O. Box 325, S-751 05 Uppsala, Sweden,
{parosh,bengt,matkin}@docs.uu.se
WWW: http://www.docs.uu.se/~{parosh,bengt,matkin}
[2] IAR Systems AB
Box 23051, 750 23 Uppsala, Sweden
mats.kindahl@iar.se
[3] Bell Laboratories, 700 Mountain Ave., Murray Hill, NJ 07974, USA
doron@research.bell-labs.com

Abstract. The purpose of partial-order reduction techniques is to avoid exploring several interleavings of independent transitions when model checking the temporal properties of a concurrent system. The purpose of symbolic verification techniques is to perform basic manipulations on sets of states rather than on individual states. We present a general method for applying partial order reductions to improve symbolic verification. The method is equally applicable to the verification of finite-state and infinite-state systems. It considers methods that check safety properties, either by forward reachability analysis or by backward reachability analysis. We base the method on the concept of commutativity (in one direction) between predicate transformers. Since the commutativity relation is not necessarily symmetric, this generalizes those existing approaches to partial order verification which are based on a symmetric dependency relation.

We show how our method can be applied to several models of infinite-state systems: systems communicating over unbounded lossy FIFO channels, and unsafe (infinite-state) Petri Nets. We show by a simple example how partial order reduction can significantly speed up symbolic backward analysis of Petri Nets.

1 Introduction

This paper is concerned with applying partial-order techniques to improve symbolic verification methods for state-space exploration.

- The purpose of *partial-order techniques* (e.g., [GP93,GW93,HP94,Pel96,Val90,Val93]) is to avoid exploring several interleavings of independent transitions, i.e., transitions whose execution order is irrelevant, e.g., because they are performed by different processes. When verifying temporal properties, partial order methods often give substantial reductions of the search space.

– The purpose of *symbolic techniques* (e.g., [BCMD92,AČJYK96,BG96]) is to perform the basic manipulations in verification on sets of states rather than on individual states. A basis is some representation of (possibly infinite) sets of states, which can be manipulated conveniently.

We use the term *constraint* to denote some representation of a set of states. Common forms of constraints are BDDs [BCMD92], zones or regions representing infinite sets of clock values of of a timed automaton [ACD90,LLPY97], upward closed sets of states of an infinite state system [AČJYK96] such as a lossy channel system [AJ93] or a Petri Net [Fin90], an infinite set of queue contents [BG96], etc. The effect of a program statement on constraints is represented by a predicate transformer. The state-space of the system is explored by generating new constraints by applying predicate transformers to already generated constraints. In this paper, we consider both forward and backward symbolic state-space exploration. To represent forward symbolic state-space exploration, we include for each statement t of the program the predicate transformer which maps a constraint φ to a constraint representing the set $post(t)(\varphi)$ of states reachable from a state in φ using statement t. To represent backward symbolic state-space exploration (e.g., as in [AČJYK96]), we include the predicate transformer which maps a constraint φ to a constraint representing the set $pre(t)(\varphi)$ of states from which a state in φ can be reached using statement t. Different exploration strategies can be used (depth-first, breadth-first, etc.). Note that standard (non-symbolic) state-based exploration corresponds to the special case where each constraint denotes a single state.

The idea in partial order techniques is to restrict the set of statements that are explored from a given state. The basis for most existing work on partial-order methods is a symmetric dependency relation on program statements, which is used to determine a subset of statements to be explored from a given state. Different criteria for reductions have been presented which are based on the concept of a dependency relation, e.g., *stubborn sets* [Val90], *persistent sets* [GP93] or *ample sets* [Pel96].

In symbolic verification, one should similarly try to reduce the set of predicate transformers that need be applied to a given constraint. As a basis for such a reduction, we have found it useful to employ the notion of *commutativity* (in one direction) between predicate transformers, originally introduced by Lipton [Lip75]. This is a weakening of the dependency relation, in that it need not be symmetric. It is more succinct than the standard definition of the symmetric dependency relation used in the literature on partial order methods.

We use commutativity to present a general definition of partial order reduction for constraint verification systems. We illustrate the applicability of our definition by

– showing that it covers our earlier work on partial order methods for symbolic verification of lossy channel systems, and
– presenting a partial order reduction on symbolic backward reachability analysis for checking the coverability problem in (unbounded) Petri Nets. To our knowledge, this is the first partial-order reduction which applies equally well

to infinite-state Petri Nets as to finite-state Petri Nets. We present a test of the reduction on a simple example.

Related Work Partial-order techniques have been employed in state-space exploration, and the literature is continuously expanding (e.g., [GP93,GW93,HP94,Pel96,Val93]). Most of this work employs a symmetric dependency relation as a basis for defining reduction strategies. Asymmetric relations are present in a few works on automated verification, e.g., [Val90]. The dependency relation can be conditional on the particular state where statements are executed [GP93,KP92]. In our formulation, the commutativity relation can be defined to be dependent on the constraint.

A combination of partial-order and symbolic techniques is also presented by Alur et al [ABH+97]. These approaches differ from ours in that they first define a partial-order reduction of the state-space, similar to the earlier literature, and thereafter explore this symbolically, using BDDs. The paper considers only forward search from initial states. In contrast, our work defines partial order reduction on top of symbolic verification, and is based on a more general assymetric commutativity relation. To our knowledge, no formulation has been given of partial order reduction for algorithms based on backward reachability analysis. Techniques for Petri Nets which exploit partial order semantics [McM95] or partial order reduction [Val90] are based on forward reachability analysis for bounded nets.

This paper is a generalization and simplification of our earlier work, [AKP97], where we considered partial-order reduction in symbolic verification of lossy channel systems [AJ93]. In this paper, we have simplified the definition of partial order reduction, and made it applicable to a range of symbolic verification methods.

Commutativity between actions or predicate transformers was introduced by Lipton [Lip75], and has been used in assertional reasoning by Back [Bac89], Lamport [Lam90], Katz and Peled [KP92], and others.

Outline. In the next section, we introduce basic definitions and constraint verification methods. In Section 3, we present our method of partial-order reductions. In Section 4, we describe how the method can be applied to symbolic verification of lossy channels and of Petri nets. We also show how partial order methods can improve verification times on a simple Petri net.

2 Programs and Symbolic Verification

We assume a program which consists of a (possibly infinite) set Σ of states, and a finite set T of transitions. Each transition $t \in T$ is a binary relation on Σ. In this paper we will consider the problem of checking reachability: given a program, a set $S_I \subseteq \Sigma$ of initial states and a set $S_F \subseteq \Sigma$, of final states, determine whether there is a sequence $\sigma_0 t_1 \sigma_1 t_2 \cdots t_n \sigma_n$ of states and transitions from some initial state $\sigma_0 \in S_I$ to some final state $\sigma_n \in S_F$ such that $\sigma_{i-1} t_i \sigma_i$ for all i with

$1 \le i \le n$. The verification of most safety properties can be transformed to the reachability problem by standard techniques.

We consider symbolic verification methods, which manipulate sets of states rather than individual states, and where the effect of transitions is represented by predicate transformers.

A *predicate* over Σ is a subset of Σ. We will often use the term *constraint* for predicates. The set of constraints over Σ forms a lattice with ordering $\sqsubseteq$ taken as set inclusion. We will say that a constraint φ *covers* another constraint φ', if $\varphi' \sqsubseteq \varphi$. Let $\bot$ be the empty constraint and let $\top$ be the set Σ.

A *predicate transformer* τ is a function from 2^Σ to 2^Σ. We will consider only predicate transformers which are monotone and strict (i.e., such that $\tau(\bot) = \bot$). Given predicate transformers $\tau_1, \ldots \tau_n$ we let $\tau_1; \ldots; \tau_n$ denote the predicate transformer τ' such that $\tau'(\varphi) = \tau_n(\cdots(\tau_1(\varphi))\cdots)$ for any constraint φ. Observe the order of application of the predicate transformers. A predicate transformer τ is *enabled* at constraint φ if $\tau(\varphi) \neq \bot$. We will consider symbolic verification algorithms which check the reachability problem either by forward reachability analysis or by backward reachability analysis. In forward analysis, we start from a set of constraints whose union is the set of initial states, and generate new constraints by applying the predicate transformer $post(t)(\varphi) = \{\sigma' \;:\; \exists \sigma \in \varphi \,.\, \sigma t \sigma'\}$ for each transition t and already generated constraint φ. New constraints which are included in already generated constraints need not be further explored. The search terminates when a constraint containing a final state is generated, or when no more constraints are generated. Backward analysis is analogous, but starts from a set of constraints that represents the set of final states, and applies the predicate transformer $pre(t)(\varphi) = \{\sigma \;:\; \exists \sigma' \in \varphi . \sigma t \sigma'\}$, attempting to find a constraint containing an initial state.

For a constraint φ, a set ψ of states, and a sequence $\rho = \tau_1; \ldots; \tau_n$ of predicate transformers in $\mathcal{T}$ we say that ρ *leads from* φ *to* ψ if $\psi \cap \tau_1; \ldots; \tau_n(\varphi) \not\equiv \bot$. We say that ψ is *reachable* from φ if there is a sequence ρ which leads from φ to ψ. We say that ψ is *reachable* from a set Φ of constraints if ψ is reachable from some $\varphi \in \Phi$. analysis to the symbolic case. **Algorithm 1** in Figure 1 is a standard generalization of state-based reachability. The algorithm repeatedly selects constraints from W to be explored. Line 4 checks if the constraint removed from W is already covered by some previously visited constraint, in which case it is redundant and can be discarded. Line 5 checks to see if we have reached a constraint that contains some final state. Line 6 computes the successor constraints to investigate. For this, a function *select* determines the subset of $\mathcal{T}$ which is to be explored from each constraint. Line 7 adds the newly processed constraint to the list of already visited constraints.

Note that at line 6, the algorithms is parameterized by a function *select* which determines the set of predicate transformers to explore from a given constraints. In the next section, we will study how the function *select* can be changed in order to reduce the search space. In this section, we will take *select*(φ) to be the set $\mathcal{T}$ of all predicate transformers. Equivalently, we can let *select*(φ) be the set *enabled*(φ) of all predicate transformers which are enabled at φ (if τ is not

Algorithm 1 (Reachability Algorithm)
Input: A finite set $\mathcal{T}$ of predicate transformers, a finite set Φ_0 of initial constraints, and a set ψ of final states.
Output: true if ψ is reachable from Φ_0.
Local Variables: A set V of constraints representing visited constraints and a working set W of constraints yet to be investigated.

```
1) Let $W = \Phi_0$ and let $V = \emptyset$
2) While $W$ is not empty, repeat steps 3–7
3)     Select and remove a constraint $\varphi \in W$
4)     If there is $\varphi' \in V$ such that $\varphi \sqsubseteq \varphi'$, goto 2
5)     If $\psi \wedge \varphi \not\equiv \bot$, then exit with the result true
6)     Add the constraints $\{\tau(\varphi) \;:\; \tau \in select(\varphi)\}$ to $W$.
7)     Add $\varphi$ to $V$
8) If $W$ becomes empty, exit with the result false
```

Fig. 1. Algorithm 1 (Reachability)

enabled at φ, then $\tau(\varphi)$ generates $\bot$ which is trivially discarded). In Section 3, we will investigate how *select* can be made even smaller, without endangering the correctness of the algorithm.

An obvious requirement on the selection of constraints in line 3 is that it is fair in the following sense: Each constraint which is inserted into W at line 6 is eventually removed at line 3 of the algorithm. We will from now on assume that any reachability algorithm under consideration satisfies this fairness condition. Breadth-first is an example of a fair exploration strategy. Depth-first need not be fair if the state-space is infinite. For the class of well-structured infinite-state systems considered in our earlier work using backward analysis [AČJYK96], any strategy is fair, since the algorithm will always terminate with W empty.

We observe that, due to the above fairness requirement, the algorithm is *complete* in the sense that it is guaranteed to return **true** if the set ψ is reachable. If ψ is not reachable. the algorithm may add an infinite sequence of constraints to W, without terminating, since the set of states and the set of constraints may both be infinite.

3 Improving the Reachability Algorithm

In this section we introduce strategies to improve the reachability algorithm presented in Section 2. The idea is to only apply a subset of the predicate transformers to a constraint, resulting in that we only explore a subset of all possible sequences of predicate transformers. Our aim is to substantially reduce the number of different constraints generated during the verification. As a basis for such a reduction, we have found it useful to employ the notion of *commutativity*, originally introduced by Lipton [Lip75].

Definition 1. *Given predicate transformers τ_2, τ_1, and a constraint φ, we say that τ_1* commutes left *with τ_2 in φ, denoted $\tau_1 \ll_\varphi \tau_2$, if $\tau_2;\tau_1(\varphi) \sqsubseteq \tau_1;\tau_2(\varphi)$.*

Intuitively, if τ_1 commutes left with τ_2 in φ, then it seems plausible that we need not apply the sequence $\tau_2;\tau_1$ to the constraint φ, since the constraint generated is a subset of the constraint generated by the sequence $\tau_1;\tau_2$.

Definition 2. *Let φ be a constraint ρ a finite sequence of predicate transformers, and τ a predicate transformer. We say that τ is* contributory *to ρ from φ if, for any partition $\rho_1;\tau';\rho_2$ of ρ we have $\tau \ll_{\rho_1(\varphi)} \tau'$.*

It is easy to see, using the definition of commutativity, that if τ is *contributory* to ρ from φ, then $\rho\tau(\varphi) \sqsubseteq \tau\rho(\varphi)$.

We will now present strategies for reducing the set of predicate transformers that need be explored from a given constraint φ. This is done in a similar spirit as for state-based partial order techniques. In our setting there are some additional differences: the search space may be infinite, and constraints are discarded if they are covered by already explored constraints. The search will be affected by the function *select*, and we are interested in requirements on *select* that guarantee completeness of the algorithm. We will present two sets of requirements, both of which guarantee completeness. The first is inspired by the definition of stubborn sets [Val90], and the second by the ample set technique [Pel96].

Our first set of requirements on *select* consists of the following two conditions.

C1 Every predicate transformer in *select*(φ) is contributory to every sequence $\rho \in (\mathcal{T} \setminus select(\varphi))^*$ of predicate transformers not in *select*(φ) from φ.

C2 There is no sequence $\rho \in (\mathcal{T} \setminus select(\varphi))^*$ of predicate transformers not in *select*(φ) which leads from φ to ψ.

Condition **C1** defines what is usually termed a persistent set [GP93]. It is the reason for why we can exploit commutativity to defer exploration of the predicate transformers outside *select*(φ). Condition **C2** is a simple way of ensuring that this deferral does not sacrifice completeness.

Theorem 1. *If the function select satisfies the conditions* **C1** *and* **C2** *for each constraint φ, then Algorithm 1 will return* **true** *if ψ is reachable from Φ_0.*

Proof. We prove by induction on n the following property: For each constraint φ generated by the algorithm which is not discarded at line 4, if some sequence $\rho \in \mathcal{T}^*$ of predicate transformers of length n leads from φ to ψ, then the algorithm will eventually generate a constraint φ' such that $\varphi' \cap \psi \neq \bot$. The case $n = 0$ is trivial. Let φ be a constraint generated by the algorithm which is not discarded, and let $\rho = \tau_1;\ldots;\tau_{n+1}$ be a sequence in $\mathcal{T}^*$ of length $n+1$ which leads from φ to ψ. By **C2**, there is an i with $1 \leq i \leq n+1$ such that $\tau_i \in select(\varphi)$. Let i be the least such i. Let $\rho_1 = \tau_i;\tau_1;\ldots;\tau_{i-1};\tau_{i+1};\ldots;\tau_{n+1}$. By **C1** and monotonicity of predicate transformers, we have $\rho(\varphi) \sqsubseteq \rho_1(\varphi)$. Thus, if $\tau_i(\varphi)$ is not discarded by the algorithm, the induction hypothesis yields the desired conclusion. If the result $\tau_i(\varphi)$ of applying τ_i to φ is discarded at line 4 because it is covered by some constraint φ_1 already in V, then by monotonicity there is a sequence of predicate transformers of length at most n which leads from the generated and not discarded constraint φ_1 to ψ, which concludes the proof of the property. Finally, letting φ be any of the initial constraints in Φ_0 proves the statement in the theorem.

We note that conditions **C1** and **C2** are both rather abstract, and that some more concrete and restrictive versions must be used in a practical implementation. For instance, condition **C2** could be enforced by finding some necessary change that must present on any sequence from φ to ψ (such as changing a particular state component), and checking that this change can only be effected by the transformers in *select*(φ).

A weaker but more complex version of Theorem 1 can be obtained by replacing **C2** by the following three conditions.

D2 If **C2** does not hold, then there is a $\tau \in$ *select*(φ) such that $\varphi' \sqcap \psi \neq \bot$ implies $\tau(\varphi') \sqcap \psi \neq \bot$ for any constraint φ'.

D3 If for some transformer $\tau \in$ *select*(φ), the successor $\tau(\varphi)$ is discarded at line 4, then *select*(φ) = *enabled*(φ) (or equivalently *select*$(\varphi) = \mathcal{T}$).

D4 If the algorithm generates an infinite path of predicate transformers, none of which is discarded, then each predicate transformer in $\mathcal{T}$ must be explored infinitely often along this path.

An explanation of these conditions should be provided by the proof of the following theorem.

Theorem 2. *If the function select satisfies the conditions* **C1**, **D2**, **D3** *and* **D4** *for each constraint φ, then Algorithm 1 will return* **true** *if ψ is reachable from Φ_0.*

Proof. Just as for Theorem 1, we prove by induction on n the following property: For each generated and not discarded constraint φ, if some transformer sequence ρ of length n leads from φ to ψ, then the algorithm will eventually generate a constraint φ' such that $\varphi' \sqcap \psi \neq \bot$. The case $n = 0$ is trivial. Let φ be a constraint generated by the algorithm which is not discarded, and let $\rho = \tau_1; \ldots; \tau_{n+1}$ be a sequence of length $n+1$ which leads from φ to ψ. If there is an i with $1 \leq i \leq n+1$ such that $\tau_i \in$ *select*(φ), we proceed as in the proof of Theorem 1. Otherwise, by condition **D2** there is a $\tau_1' \in$ *select*(φ) such that $\tau_1; \ldots;; \tau_{n+1}; \tau_1'(\varphi) \sqcap \psi \neq \bot$. It follows by **C1**, falsity of **C2**, and monotonicity that $\tau_1'; \tau_1; \ldots;; \tau_{n+1}(\varphi) \sqcap \psi \neq \bot$. Let $\varphi_1 = \tau_1'(\varphi)$. By condition **D3**, we infer that τ_1' is not discarded (remember that τ_1 is not explored from φ). Just as for φ we have that ρ leads from φ_1 to ψ. If there is an i with $1 \leq i \leq n+1$ such that $\tau_i \in$ *select*(φ_1), we proceed as in the proof of Theorem 1. Otherwise, we proceed as with φ. In this way, we generate a path $\varphi\varphi_1\varphi_2 \cdots$ of generated and not discarded constraints. The path stops if at some point condition **C2** holds, in which case we are done. If not, the path is infinite. By condition **D4** there is a j such that $\tau_1 \in$ *select*(φ_j). Regardless of whether $\tau_1(\varphi_j)$ is discarded or not, we can use the induction hypothesis to conclude the proof, just as in the previous theorem.

4 Examples

In this section, we apply the results in the previous section to some models of infinite state systems.

4.1 Lossy Channel Systems

A *lossy channel system* consists of a *control part* and a *channel part*. The control part is modeled as a number of finite-state processes communicating via the channels, while the channel part consists of a finite set of channels. Each channel behaves as a FIFO buffer which is unbounded and unreliable in the sense that it can lose messages. A channel is used to perform asynchronous communication between a pair of processes, so for each channel there is unique process sending messages to the channel, and a unique process receiving messages from the channel. A constraint defines the control state of each process, and defines for each channel the contents of the channel as an upward closed set of strings. Predicate transformers correspond to backward performing of send and receive transitions. In [AKP97] we apply a partial order technique for a symbolic backward reachability analysis described in [AJ93]. An algorithm (satisfying **C1** - **C2**) for computing *select* at a particular configuration is given as follows. If there is a process where all its enabled transitions are either send transitions to a non-empty channel, or receive transitions, then select exactly all enabled transitions of the process. If no such process exist, then select the set of enabled transitions of all processes. In [AKP97] we apply the above algorithm to a *go back n* protocol, obtaining time reductions of up to 25%, and to a mutual exclusion protocol, obtaining time reductions of up to 97%.

4.2 Petri Nets

In this section we will apply the techniques to algorithms for checking the coverability problem for Petri Nets. A Petri net is a tuple $\mathcal{N} = \langle P, T, in, out\rangle$, where P is a finite set of *places*, T a finite set of *transitions*, and $in : T \mapsto (P \mapsto \mathcal{N})$ and $out : T \mapsto (P \mapsto \mathcal{N})$ are functions that for each transition $t \in T$ define how many tokens are consumed and produced at each place when t is fired. A *marking* m is a mapping from P to $\mathcal{N}$. A transition t can fire in a marking m if $in(t)(p) \leq m(p)$ for each place $p \in P$. When t fires, the marking is changed from m to the marking m' defined by $m'(p) = m(p) - in(t)(p) + out(t)(p)$ for each $p \in P$. We define $\ominus$ (monus) by $a \ominus b = \max(0, a - b)$. We define the operators $+$ and $-$ on markings in the natural way, and the partial order $\leq$ by point-wise extension, i.e. $m_1 \leq m_2$ iff $m_1(p) \leq m_2(p)$ for every place $p \in P$.

A set φ of markings is *upward closed* if $m \in \varphi$ implies $m' \in \varphi$ for all m' with $m \leq m'$. We will be interested in the coverability problem, which is defined as follows: Given a set M_I of initial markings, and a set M_F of final markings, determine whether there is a set of transitions leading from a marking in M_I to a marking m which covers some marking $m_F \in M_F$ in the sense that $m_F \leq m$. The coverability problem can be checked by backward search as follows. Let our constraints be sets of the form $\varphi_{m_0} = \{m \ : \ m_0 \leq m\}$ for some m_0. Note that $pre(t)(\varphi_m) = \varphi_{m \ominus out(t) + in(t)}$. In **Algorithm 1**, let Φ_0 be a finite set of constraints, whose union is M_F, and let the set $\mathcal{T}$ be the set $\{pre(t) \ : \ t \in T\}$. Let ψ be M_I. **Algorithm 1** then represents a symbolic backward analysis for solving the coverability problem in the case that $select(\varphi) = \mathcal{T}$ for each φ. It can be shown (e.g., [AČJYK96]) that this analysis is guaranteed to terminate.

We can now present a partial order reduction strategy which is based on sufficient (but in general not necessary) criteria for commutativity.

We say that the predicate transformers $pre(t_1)$ and $pre(t_2)$ are in *conflict* if for some place p we have $out(t_1)(p) > 0$ and $out(t_2)(p) > 0$. We say that a transformer $pre(t)$ is *deficient* for p at φ_m if $m(p) < out(t)(p)$. We say that a transformer $pre(t)$ *separates* a marking m from a set M_I of markings if there is a place p with $out(t)(p) > 0$ such that $m(p) > m_I(p)$ for each marking $m_I \in M_I$. We observe that if $pre(t)$ separates m from M_I then any sequence of predicate transformers that leads from φ_m to some φ_{m_I} with $m_I \in M_I$ must contain a predicate transformer $pre(t')$ for a transition with $out(t')(p) > 0$, where p is the place that makes t' separating. It follows that the sequence must contain either $pre(t)$ or a transformer which is in conflict with $pre(t)$.

The following is a procedure to generate a set of predicate transformers $select(\varphi_m)$ to be explored from a constraint φ_m. We assume that the set M_I (corresponding to ψ in Algorithm 1) is given.

- Start with some transformer $pre(t_0)$ which separates m from M_I.
  Let $select(\varphi_m) = \{pre(t_0)\}$ initially.
  Repeatedly add to $select(\varphi_m)$ all transformers $pre(t')$ for which there is a transformer $pre(t) \in select(\varphi_m)$ such that either
  (1) $pre(t)$ and $pre(t')$ are in conflict, or
  (2) there is a place p such that $pre(t)$ is deficient at p and $in(t')(p) > 0$
- If no transformer $pre(t_0)$ can be found which separates m from M_I, let $select(\varphi_m) = \mathcal{T}$, i.e., explore all transformers.

It can be checked that this procedure generates a subset of transitions (i.e., predicate transformers) which satisfies conditions **C1** and **C2**. An intuitive explanation is as follows. By the observation above, if $pre(t_0)$ separates m from M_I then any sequence of predicate transformers that leads from φ_m to some φ_{m_I} with $m_I \in M_I$ must contain a transition in $select(\varphi)$: either $pre(t_0)$ or a transformer which is in conflict with $pre(t_0)$. Thus **C2** is satisfied. To verify **C1**, first note that conditions (1) and (2) imply that $pre(t) \ll_{\varphi_m} pre(t')$ for all $pre(t) \in select(\varphi_m)$ and $pre(t') \notin select(\varphi_m)$. Suppose that after a sequence of transformers we reach a constraint $\varphi_{m'}$ where $pre(t) \not\ll_{\varphi_{m'}} pre(t')$ but $pre(t) \ll_{\varphi_m} pre(t')$. Then the sequence must remove tokens from a place p with $out(t)(p) > 0$ and hence contain a transition which is in conflict with $pre(t)$.

Measurements. we will describe the results from a small experiment with the procedure for generating reduced sets of predicate transformers at the end of Section 4.2 following heuristics to select ample sets.

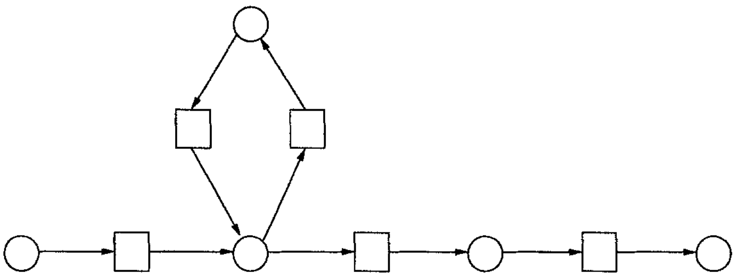

Fig. 2. A Petri Net Buffer

The example we used consists of a Petri net that moves tokens from an initial place (to the left) to a final place (to the right). We also added a loop to the sequence to show that a Petri net which is not a simple directed graph can be handled. The net can be seen in figure 2. The results from executing this Petri net is shown in the following table. The column "Tokens" denotes the number of tokens in the final place at the start of the backward analysis. The running times are in seconds.

Tokens	Standard Algorithm	Partial Order Algorithm
4	0.1	0.0
8	1.5	0.1
12	13.3	0.2
16	78.8	0.3
20	346.3	0.6
24	1239.7	0.9
28	–	1.5
32	–	2.3
36	–	3.3
40	–	4.6

As can be seen, the standard method degenerates very fast, while the partial order method is still well within acceptable running times. The good performance comes from the fact that, using our method for selecting reduced sets of transformers, the partial order method will move one token at a time from the end place to the initial place. The standard algorithm will investigate every interleaving of moving tokens from the end place closer to the start place.

References

[ABH+97] R. Alur, R.K. Brayton, T.A. Henzinger, S. Qadeer, and S.K. Rajamani. Partial-order reduction in symbolic state space exploration. In O. Grumberg, editor, *Proc. 9th Int. Conf. on Computer Aided Verification*, volume 1254, pages 340–351, Haifa, Israel, 1997. Springer Verlag.

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th IEEE Int. Symp. on Logic in Computer Science*, pages 414–425, Philadelphia, 1990.

[AČJYK96] Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Tsay Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. 11th IEEE Int. Symp. on Logic in Computer Science*, pages 313–321, 1996.

[AJ93] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *Proc. 8th IEEE Int. Symp. on Logic in Computer Science*, pages 160–170, 1993.

[AKP97] Parosh Aziz Abdulla, Mats Kindahl, and Doron Peled. An improved search strategy for Lossy Channel Systems. In Tadanori Mizuno, Nori Shiratori, Teruo Hegashino, and Atsushi Togashi, editors, *FORTE X / PSTV XVII '97*, pages 251–264. Chapman and Hall, 1997.

[Bac89] R.J.R. Back. A method for refining atomicity in parallel algorithms. In *Proc. PARLE 89*, volume 366 of *Lecture Notes in Computer Science*, pages 199–216. Springer Verlag, 1989.

[BCMD92] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In Alur and Henzinger, editors, *Proc. 8th Int. Conf. on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, 1996.

[Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89:144–179, 1990.

[GP93] P. Godefroid, D. Pirottin. *Refining Dependencies Improves Partial-Order Verification Methods.* Proc. 5th Conference on Computer Aided Verification, Lecture Notes in Computer Science 697, Springer, 438–449. Elounda, Greece, 1993.

[GW93] P. Godefroid and P. Wolper. Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties. In *Formal Methods in System Design*, Kluwer, 2 (1993), 149–164.

[HP94] G.J. Holzmann and D. Peled. An improvement in formal verification. In *Proc. FORTE '94*, pages 197–211, 1994.

[JZ92] W. Janssen and J. Zwiers. From sequential layers to distributed processes. In *Proc. 11th ACM Symp. on Principles of Distributed Computing, Canada*, 1992.

[KP92] Shmuel Katz and Doron Peled. Defining conditional independence using collapses. *Theoretical Computer Science*, 101:337–359, 1992.

[KLM+98] R.P. Kurshan, V. Levin, M. Minea and D. Peled, H. Yenigun,. Static Partial Order Reduction. *TACAS'98, Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Lisbon, Portugal.

[Lam90] L. Lamport. A theorem on atomicity in distributed algorithms. *Distributed Computing*, 4(2):59–68, 1990.

[Lip75] Lipton. Reduction, a method of proving properties of parallel programs. *Communications of the ACM*, 18(12):717–721, Dec. 1975.

[LLPY97] K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proc. 18th IEEE Real-Time Systems Symposium*, pages 14–24, San Francisco, California, Dec. 1997.

[McM95] K.L. McMillan. A technique of a state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[Pel96] D. Peled. Combining Partial Order Reductions with On-the-fly Model-Checking. *Journal of Formal Methods in Systems Design*, 8 (1996), 39–64.

[Val90] A. Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets*, number 483 in Lecture Notes in Computer Science, pages 491–515. Springer-Verlag, 1990.

[Val93] A. Valmari. On-the-fly verification with stubborn sets. In Courcoubetis, editor, *Proc.* 5^{th} *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, pages 59–70, 1993.