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

Parosh Aziz Abdulla* Bengt Jonsson† Mats Kindahl‡
Doron Peled§

Abstract

The purpose of partial-order verification techniques is to avoid exploring several interleavings of independent transitions. 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.

*e-mail adress is `parosh@docs.uu.se`
†e-mail adress is `doron@research.bell-labs.com`
‡e-mail adress is `matkin@docs.uu.se`
§e-mail adress is `bengt@docs.uu.se`

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., [God96, GW90, HP96, Pel93, 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 safety 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 represent the symbolic verification of a particular system by a *constraint verification system*. This consists of a (possibly infinite) set of constraints (predicates), each representing a (possibly infinite) set of states, and a finite set of predicate transformers. A predicate transformer represents the effect of a program statement on constraints. 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 state-space of the system is explored by generating new constraints by applying predicate transformers to already generated constraints. Constraint verification systems can represent several existing variants of symbolic verification. 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. With a given constraint verification system, different exploration strategies can be used (depth-first, breadth-first, etc.). Note that standard (non-symbolic) state-based exploration can also be represented as a constraint verification system, by letting each constraint denote a single state.

The idea in partial order techniques is to restrict the set of statements that are explored from a given state. In our case, we should seek to apply only a small subset of all predicate transformers to a given constraint. The basis for most existing work on partial-order methods is a symmetric dependency relation on program statements. This symmetric relation is used to determine a subset of statements to be explored at a given state.

When lifting this work to symbolic verification, 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 should be noted that the definition of commutativity is very simple. 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., [God96, GW90, HP96, Pel93, 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 [GW90]. 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]. Their approach differs 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.

Commutatitivity between actions or predicate transformers was introduced by Lipton [Lip75], and has been used by Back [Bac89], Lamport [Lam90], Katz and Peled

[KP92], and others. It is therefore possible that this paper could lead to a bridge between partial order techniques in assertional verification (e.g., [JZ92]) and in automated verification.

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. In Section 5 we 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. We will in this paper 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 $1 \leq i \leq n$. 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. Let $\perp$ 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 $\tau(\perp) = \perp$). 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 $\tau(\varphi) \neq \perp$. 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 $post(t)(\varphi) = \{ \sigma' \mid \exists \sigma \in \varphi \,.\, \sigma t \sigma' \}$ 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 $pre(t)(\varphi) = \{ \sigma \mid \exists \sigma' \in \varphi . \sigma t \sigma' \}$, attempting to find a constraint containing an initial state.

We introduce *constraint verification systems* as a uniform formulation of these symbolic methods for checking the reachability problem. A constraint verification system consists of a set of constraints and a finite set $\mathcal{T}$ of predicate transformers. In forward analysis the predicate transformers are $post(t)$ for each transition t, and in backward analysis they are $pre(t)$ for each transition $t \in T$. This formulation covers finite-state reachability verification, reachability analysis for e.g., timed automata, Petri nets, systems of Finite state Machines communicating over unbounded channels

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.
Algorithm 1 is a description of how reachability analysis can be carried out, given a constraint verification system.

Algorithm 1 (Reachability Algorithm)
Input: A constraint verification system with 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 some $\varphi_0 \in \Phi_0$.
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 $\{ \tau(\varphi) \mid \tau \in select(\varphi) \}$ to W.
7)   Add φ to V
8) If W becomes empty, exit with the result **false**

The algorithm repeatedly selects constraints from W to be explored. Line 4 checks if the constraint removed from W is already covered by the previously visited constraints, 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.

We will study how the function *select* can be changed in order to reduce the search space. Naively, $select(\varphi)$ can be taken to be the set $\mathcal{T}$ of all predicate transformers. A slightly less naive choice is to let $select(\varphi)$ be the set of all predicate transformers which are enabled at φ. In Section 3, we will investigate how *select* can be made even smaller, without endangering the correctness of the algorithm.

We note that since the set of states may in general be infinite, the above algorithm need not terminate. Termination may depend on the order in which constraints are explored. We say that an exploration strategy is *fair* if the following condition holds: If a constraint is at some point in W, then it is eventually removed at line 3 of the algorithm. We see that under a fair exploration strategy, the algorithm will always return **true** if the set ψ is reachable. 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 [AČJYK96] using backward analysis any strategy is fair, since the algorithm will always terminate with W empty.

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. The hope is to substantially reduce the number of different constraints generated during the verification.

Definition 3.1. Given predicate transformers τ_1, τ_2, and predicate φ, we say that τ_2 *commutes left* with τ_1 in φ, denoted $\tau_2 \ll_\varphi \tau_1$, if $\tau_1;\tau_2(\varphi) \sqsubseteq \tau_2;\tau_1(\varphi)$.

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

Definition 3.2. Let φ be a predicate, 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, where τ' is a predicate transformer, 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 entail already explored constraints. The search will be affected by the function *select*, and by the particular exploration strategy used.

Theorem 3.3. *If the function select satisfies the below conditions* **B1** *and* **B2** *for each constraint φ, and if the exploration strategy is fair, then Algorithm 1 will return* **true** *if ψ is reachable from some $\varphi_0 \in \Phi_0$.*

B1 *Every predicate transformer in $select(\varphi)$ is contributory to to every sequence $\rho \in (\mathcal{T} \setminus select(\varphi))^*$ of predicate transformers not in $select(\varphi)$ from φ.*

B2 *There is no sequence $\rho \in (\mathcal{T} \setminus select(\varphi))^*$ of predicate transformers not in $select(\varphi)$ which leads from φ to ψ.*

Proof. (Sketch) We prove by induction on n the following property: For each constraint φ generated in the algorithm and not discarded, 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' in such that $\varphi' \cap \psi \neq \perp$. 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 **B2**, 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 **B1** and monotonicity of predicate transformers, we have $\rho(\varphi_0) \sqsubseteq \rho_1(\varphi_0)$. Thus, if $\tau_i(\varphi)$ is not discarded by the algorithm, the induction hypothesis yields the desired conclusion since there is a sequence of predicate transformers of length n which leads from $\tau_i(\varphi)$ to ψ. If the result $\tau_i(\varphi)$ of applying τ_i to φ is discarded because it entails a predicate φ' already in V, then by monotonicity there is a sequence of predicate transformers of length at most n which leads from φ' to ψ, and we are done.

We note that a more general version of Theorem 3.3 can be obtained by replacing **B2** by the following three conditions.

C2 If there is a sequence $\rho \in (\mathcal{T} \setminus select(\varphi))^*$ of predicate transformers not in $select(\varphi)$ which leads from φ to ψ, then there is a $\tau \in select(\varphi)$ which satisfies the property that $\varphi' \cap \psi \neq \perp$ implies $\tau(\varphi') \cap \psi \neq \perp$ for any constraint φ'.

C3 If for some node φ and some predicate transformer $\tau \in select(\phi)$, the potential successor $\tau(\varphi)$ of φ is discarded, then $select(\phi) = \mathcal{T}$.

C4 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.

A proof is omitted in this extended abstract.

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 modelled 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 with a symbolic backward reachability analysis described in [AJ93]. An algorithm (satisfying **B1** - **B2**) for computing choosing *select* at a particular configuration is given as follows. If there is a process where all enabled transitions of the process are either send transitions to a non-empty channel, or receive transitions, then select exactly all enabled transitions of the process. If no such a process exists take the ample set to be 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

We will in this section 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 $in : T \mapsto (P \mapsto \mathbb{N})$ and $out : T \mapsto (P \mapsto \mathbb{N})$ 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 $\mathbb{N}$. A transition t can fire in 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 $\varphi_{m_0} = \{ m \mid m_0 \leq m \}$ for some m_0. In Algorithm1, let Φ_0 be a finite set of constraints, whose union is M_F. Let the set $\mathcal{T}$ be the set $\{ pre(t) \mid t \in T \}$, and 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.

We note that $pre(t, \varphi_m) = \varphi_{m \ominus out(t) + in(t)}$. In the following, we shall abuse notation and use transitions t to actually denote their corresponding predicate transformer $pre(t)$. Thus $t(\varphi_m) = \varphi_{m \ominus out(t) + in(t)}$. We can actually calculate a closed expression for when $t_1 \ll_{\varphi_m} t_2$. We will here present a partial order reduction strategy which is based on stronger but sufficient criteria for commutativity.

We say that t_1 and 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 transition t is *deficient* for p at φ_m if $m(p) < out(t)(p)$. We say that a transition 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 $m_I \in M_I$.

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

It can be checked that this procedure generates a subset of transitions (i.e., predicate transformers) which satisfies conditions **B1** and **B2**. An intuitive explanation is as follows. If t_0 separates m from M_I then let p_0 be such that $out(t_0)(p_0) > 0$ and such that $m(p_0) > m_I(p_0)$ for each initial marking $m_I \in M_I$. It follows that **B2** is satisfied, since in order to reach a marking in M_I from m we must perform a backwards transition which decreases the number of tokens in p_0, i.e., a transition t with $out(t)(p_0) > 0$. However, all such transitions t are in conflict with t_0, hence in $select(\varphi_m)$. Furthermore, if $t \not\ll_{\varphi_m} t'$ then t and t' must satisfy condition 1. or 2. above. Hence $t \ll_{\varphi_m} t'$ for all $t \in select(\varphi_m)$ and $t' \notin select(\varphi_m)$. To verify **B1**, suppose that after a sequence of transitions we reach a constraint $\varphi_{m'}$ where $t \not\ll_{\varphi_{m'}} t''$ but $t \ll_{\varphi_m} t''$. 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 t.

5 Measurements

In this section we will describe the results from experiments with the procedure for generating reduced sets of predicate transformers at the end of Section 4.2 following heuristics to select ample sets.

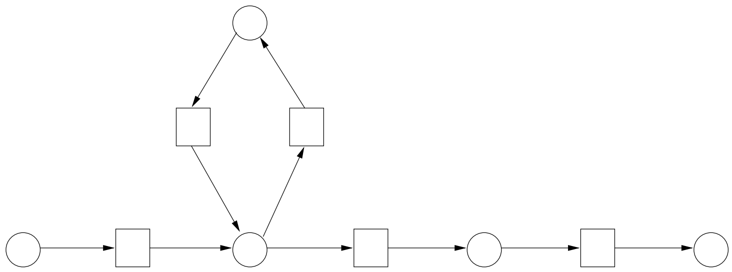

Figure 1: A Petri Net Buffer

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 have 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 1. The results from executing this Petri net is shown in the following table. The column "Tokens" denote 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 preformance 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. 9^{th} 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. 5^{th} 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. 11^{th} 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. 8^{th} 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. 8^{th} 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.

[God96] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem.* PhD thesis, University of Liège, 1996. vol. 1032 of Lecture notes in Computer Science, Springer Verlag.

[GW90] P. Godefroid and P. Wolper. Using partial orders to improve automatic verification methods. In *Proc. Workshop on Computer Aided Verification*, 1990.

[HP96] G.J. Holzmann and D.A. Peled. An improvement in formal verification. In *Proc. FORTE '94*, pages 197–211, 1996.

[JZ92] W. Janssen and J. Zwiers. From sequential layers to distributed processes. In *Proc. 11^{th} 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.

[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. 18^{th} 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.

[Pel93] D. Peled. All from one, one for all, on model-checking using representatives. In *Proc. 5^{th} Int. Conf. on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, 1993.

[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.