Partial-Order Reduction in Symbolic State Space Exploration*

R. Alur R.K. Brayton T.A. Henzinger S. Qadeer S.K. Rajamani

EECS Department, University of California, Berkeley, CA 94720, U.S.A.
Email: {alur, brayton, tah, shaz, sriramr}@eecs.berkeley.edu

Abstract. State space explosion is a fundamental obstacle in formal verification of designs and protocols. Several techniques for combating this problem have emerged in the past few years, among which two are significant: partial-order reductions and symbolic state space search. In asynchronous systems, interleavings of independent concurrent events are equivalent, and only a representative interleaving needs to be explored to verify local properties. Partial-order methods exploit this redundancy and visit only a subset of the reachable states. Symbolic techniques, on the other hand, capture the transition relation of a system and the set of reachable states as boolean functions. In many cases, these functions can be represented compactly using binary decision diagrams (BDDs). Traditionally, the two techniques have been practiced by two different schools—partial-order methods with enumerative depth-first search for the analysis of asynchronous network protocols, and symbolic breadth-first search for the analysis of synchronous hardware designs. We combine both approaches and develop a method for using partial-order reduction techniques in symbolic BDD-based invariant checking. We present theoretical results to prove the correctness of the method, and experimental results to demonstrate its efficacy.

1 Introduction

The number of states of a system grows exponentially with the length of its description. Commonly known as *state space explosion*, this is a fundamental problem in formal verification of systems like hardware designs, network protocols, and distributed algorithms. Techniques like model checking of temporal-logic properties, which depend on state space exploration, have been limited in their use because of this problem. A variety of heuristics have been proposed to combat state space explosion. Two significant solutions that are supported by existing tools are *symbolic model checking* and *partial-order reduction*:

- In synchronous hardware designs, state space explosion manifests itself in the number of states growing exponentially with the number of state variables in the design. Symbolic model checking [BCMD92, McM93] avoids explicit construction of the state space. The transition relation of the system and state sets are modeled as boolean functions and represented using binary decision diagrams (BDDs). Model checking of temporal-logic properties, then, reduces to symbolic fixpoint computation that uses BDD-based image computation as a primitive.
- In asynchronous systems consisting of a set of communicating concurrent processes (for example, network protocols and distributed algorithms), the behavior of the system can be

* This research was supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the AFOSR contract F49620-93-1-0056, by the ARO MURI grant DAAH-04-96-1-0341, by the ARPA grant NAG2-892, and by the Semiconductor Research Corporation contracts DC-324.036 and DC-324.005.

modeled as the set of all interleavings of the events in individual processes. One source of state space explosion are the $n!$ possible interleavings for n concurrent events. If the concurrent events are independent, then all interleavings are equivalent in that they lead to the same state. Partial-order semantics described in [Maz88] for the composition of concurrent processes group equivalent interleavings of concurrent events together into *traces*. It has been shown that sometimes it suffices to explore just one representative interleaving from each trace for verifying temporal-logic properties [Val91, Pel93, GW94]. Consequently, during state space exploration, in each state it suffices to search an *ample* subset of the enabled transitions, rather than all of them, leading to significant reduction in the explored state space for some asynchronous protocols [HP94, God96].

In this work, we address the problem of *combining* partial-order reduction methods with symbolic state space traversal for invariant checking. Such a combination is particularly useful in cases where the number of states that have to be explored remains exponential in the size of the system description, even after partial-order reduction. Consider, for example, a leader-election protocol, where the initial states of the processes are unknown. The correctness criterion asserts that *irrespective of the initial states* of the processes, the desired outcome must be ensured. Since the number of initial states is exponential in the number of processes, explicit state space search is forced to explore an exponential number of states, even with partial-order reduction. Partial-order reductions, however, give flexibility in the sets of states that need to be represented in symbolic search. Instead of computing images with the whole transition relation, it suffices to compute symbolically an ample set of successor states, or any set of successor states that contains an ample set. This flexibility adds a new dimension to heuristic techniques like modified search order [BCL91] and use of don't cares [RAP$^+$95], in symbolic state space exploration.

The following toy example illustrates the advantage of combining symbolic and partial-order methods. Consider n processes $P_1, P_2, \ldots, P_n$. Each process P_i increments a local variable x_i independent of all the other processes, until $x_i = N$, after which x_i remains equal to N. Each of the x_i's is initialized to 0. We assume interleaving semantics for process composition, that is, with each transition, only one process updates its local variable. There are n transitions out of each global state in which $x_i \neq N$ for all i. For the purpose of comparing symbolic and enumerative approaches we assume, without loss of generality, that the variable ordering is $x_1, x_2, \ldots, x_n$. A naive enumerative algorithm has to explore $O(N^n)$ states. If we perform symbolic search using MDDs[2] (multi-valued decision diagrams) on this example, the set of reached states after k steps ($k \leq N$) is defined by the relation $\sum_{i=1}^{n} x_i \leq k$, whose MDD representation has size $\Theta(nk)$. Suppose we are interested in checking a local invariant, such as $x_3 \leq N$. For checking such an invariant, it is not necessary to explore all transitions out of a state. In fact, using partial-order reduction, it suffices to first increment x_1 until $x_1 = N$, then increment x_2 until $x_2 = N$, and so on. Thus, the number of explored states is reduced to $O(Nn)$. Both methods improve significantly on naive state space exploration.

Moreover, if symbolic search is performed on the partial-order reduced state space, the MDD representation of the set of reached states after any number of steps has size $\Theta(n)$, which is an improvement over either method in isolation. This improvement becomes even more significant if the variables have nondeterministic initial values. Suppose that each x_i is initialized nondeterministically to either 0 or 1. An enumerative algorithm has to visit 2^n initial

[2] MDDs are a generalization of BDDs in which the variables can take on values from any finite set instead of just $\{0,1\}$.

states, even with partial-order reduction.[3] In symbolic search, the set of reached states after k steps ($k \leq N$) is defined by the relation $\sum_{i=1}^{n} \langle x_i - 1 \rangle \leq k$, where $\langle x \rangle$ stands for x if $x \geq 0$, and for 0 otherwise. It can be shown that the size of the MDD representing the above relation is $\Omega(n^2 + nk)$. However, for symbolic state space traversal using the partial-order reduced transition relation, the MDD representing the set of reached states after any number of steps is only $\Theta(n)$. Similar improvements in search efficiency are obtained in two examples in Section 4—a leader-election protocol in a unidirectional ring and an asynchronous tree arbiter circuit.

Usually, partial-order reductions are implemented using a modified depth-first search algorithm. In Section 2, we show that correct partial-order reductions can be obtained on any search technique that has a certain property. Breadth-first search is shown to have this property. Consequently, partial-order reductions can be applied to symbolic search (which is inherently breadth-first). Partial-order methods explore only a subset of the enabled transitions at each state, called an *ample set*. For the automatic implementation of our method, ample sets have to be computed automatically in a symbolic setting. In Section 3, we present a symbolic version of a standard algorithm [God96] for computing ample sets.

2 Partial-Order Reduction for Breadth-First Search

In [Val91, Pel93, GW94], partial-order reductions are obtained by a modified depth-first search of the state space. We generalize their results to obtain a modified breadth-first search algorithm, which can then be used to explore the reduced state space symbolically. A similar generalization is also obtained by [CP96] in the context of mechanically verifying the partial-order reduction techniques implemented in the model checker SPIN [HP94].

A *labeled transition graph* is a 5-tuple $G = \langle V, S, \hat{S}, \Gamma, \rightarrow \rangle$ with the following components:

- A finite set V of binary *variables*.
- The set $S = 2^V$ of all possible valuations for the variables, called the *state set*.
- A set of *initial states* $\hat{S} \subseteq S$.
- A finite set Γ of *actions*.
- A *transition relation* $\rightarrow \subseteq S \times \Gamma \times S$, with the restriction that if (s, α, t) and (s, α, t') are in $\rightarrow$, then $t = t'$.

We write $s \xrightarrow{\alpha} t$ if $(s, \alpha, t) \in \rightarrow$, and $s \rightarrow t$ if there exists an action α such that $s \xrightarrow{\alpha} t$. The set of actions *enabled* in state s is defined as $enabled(s) = \{\alpha \mid \exists t.\ s \xrightarrow{\alpha} t\}$. An action sequence $\overline{\alpha} = \alpha_1 \alpha_2 \ldots \alpha_m$ in Γ^* is G-*enabled* in state s if there exist states $s_1, \ldots, s_{m-1}, t$ such that $s \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{m-1}} s_{m-1} \xrightarrow{\alpha_m} t$. In this case, the state t is denoted by $succ(s, \overline{\alpha})$ and called the $\overline{\alpha}$-*successor* of s.

The algorithm to obtain a partial-order reduction performs a selective search. While exploring the successors of a state s, instead of considering all actions enabled in s, the algorithm chooses only a subset of the enabled actions, and thus, constructs a subgraph of the original graph. Let $G = \langle V, S, \hat{S}, \Gamma, \rightarrow \rangle$ be a labeled transition graph. A graph $G' = \langle V, S', \hat{S}, \Gamma, \rightarrow' \rangle$ is a subgraph of G if $S' \subseteq S$ and $\rightarrow' \subseteq \rightarrow$. A selective search can be specified by a function from states to subsets of actions. For a function Δ from S to 2^Γ, the subgraph

[3] In this particular example, since the processes are initialized independently and the behavior of the system from different initial states is identical, the system description could be modified so that enumerative search with partial-order reduction is efficient. But such a simplification is not possible in the general case.

$G_\Delta = \langle V, S_\Delta, \hat{S}, \Gamma, \rightarrow_\Delta \rangle$ of G is a *reduction* of G with respect to Δ if G_Δ is the smallest graph such that (1) $\hat{S} \subseteq S_\Delta$ and (2) for every state $s \in S_\Delta$, if $s \xrightarrow{\alpha} t$ for some $\alpha \in \Delta(s)$, then $s \xrightarrow{\alpha}_\Delta t$ and $t \in S_\Delta$. In the remainder of this section, we develop requirements on Δ so that the reduction G_Δ can be used for the verification of certain invariants.

Partial-order equivalence We proceed to define an equivalence relation over the action sequences of a labeled transition graph so that, when solving certain reachability problems, it suffices to explore only one representative from each equivalence class. The equivalence is defined using the notion of independence between actions.

Let α and β be two different actions of G. The two actions α and β are *independent*, written $\alpha \sim \beta$, if for all states $s \in S$, (1) if α is enabled in s, then β is enabled in s iff β is enabled in $succ(s, \alpha)$, (2) if β is enabled in s then, α is enabled in s iff α is enabled in $succ(s, \beta)$, and (3) if both α and β are enabled in s, then $succ(s, \alpha\beta) = succ(s, \beta\alpha)$. Intuitively, two actions are independent if they neither enable nor disable each other, and the order in which the two actions are executed does not affect the state that is reached. For every labeled transition graph G, the independence relation $\sim$ on the actions of G is irreflexive and symmetric.

For two action sequences $\overline{\gamma}$ and $\overline{\gamma}'$ of a labeled transition graph G, define $\overline{\gamma} \equiv \overline{\gamma}'$ if $\overline{\gamma} = \overline{\gamma_1}\alpha\beta\overline{\gamma_2}$ and $\overline{\gamma}' = \overline{\gamma_1}\beta\alpha\overline{\gamma_2}$, with $\alpha \sim \beta$. The *partial-order equivalence* $\equiv^*$ on the action sequences of G is the reflexive-transitive closure of $\equiv$. Intuitively, two action sequences are partial-order equivalent if one can be obtained from the other by repeatedly commuting adjacent independent actions. The set of all action sequences partial-order equivalent to an action sequence $\overline{\alpha}$ is represented by $[\overline{\alpha}]$. Such an equivalence class is called a Mazurkiewicz trace [Maz88].

Persistent functions The actions in the subset of actions selected to be explored from a state should be independent, not only of all the remaining actions enabled in state s, but also independent of all actions enabled in a state reachable from s by executing other actions. Such a subset of actions is called persistent. Formally, a function Δ from S to 2^Γ is *persistent* if for every state s of G the following holds: for all actions $\alpha \in \Delta(s)$, (1) α is enabled in s, and (2) if there is an action sequence $\overline{\beta} = \beta_1\beta_2 \ldots \beta_n \in (\Gamma \backslash \Delta(s))^*$ that is G-enabled in s, then α is independent of every action β_i in $\overline{\beta}$. We say that $\Delta(s)$ is a *persistent set* of actions at s.

A selective search based on exploring, from every state, only a persistent set of actions needs to ensure that an action is not delayed forever. Hence, if the persistent set at a state does not lead to a new state during the state space exploration, then all enabled actions need to be explored. Given a persistent function Δ, a mapping Ψ from the states of the reduction G_Δ to $\mathbb{N}$ is a *witness* for Δ if for all states s, if $\Delta(s) \neq enabled(s)$, then there exists an action $\alpha \in \Delta(s)$ such that $\Psi(succ(s, \alpha)) < \Psi(s)$. The well-foundedness of Ψ ensures that an action is not ignored everywhere in a loop. A *partial-order reduction* of G is a reduction of G with respect to a persistent function that has a witness.

Let $\overline{\alpha}$ and $\overline{\beta}$ be two action sequences. We say that $\overline{\beta}$ *subsumes* $\overline{\alpha}$ if there exists an action sequence $\overline{\gamma}$ such that $\overline{\alpha\gamma} \equiv^* \overline{\beta}$. The relationship between a graph and its partial-order reduction is captured by the following theorem, which states that every action sequence enabled in an initial state of the original graph is subsumed by some sequence enabled in the reduction.

Theorem 1. *Let s be a state in a partial-order reduction G_Δ of a labeled transition graph G, and let $\overline{\alpha}$ be an action sequence G-enabled in s. There exists an action sequence $\overline{\beta}$ that is G_Δ-enabled in s such that $\overline{\alpha}$ is subsumed by $\overline{\beta}$.*

$\sigma_{new} := \hat{S}$
repeat
  /* σ is the current set of reached states */
  $\sigma := \sigma_{new}$
  Frontier $\mathcal{F} := \{s \in \sigma \mid \exists t \notin \sigma.\ s \rightarrow t\}$
  Let $\Delta(s)$ be an ample set at s with respect to the history $\tau_B(s)$, for every $s \in \mathcal{F}$
  $\sigma_{next} := \cup_{s \in \mathcal{F}} \{t \mid \exists \alpha \in \Delta(s).\ s \xrightarrow{\alpha} t\}$
  $\sigma_{new} := \sigma \cup \sigma_{next}$
until $\sigma = \sigma_{new}$

Fig. 1. Modified breadth-first search algorithm

Ample functions To compute partial-order reductions, we need persistent functions that can be shown to have witnesses. For this purpose, we consider a special class of persistent functions, called ample functions, which facilitate the construction of witnesses by taking into account the set of states visited so far during the selective search. A *history function* τ is a map from S to 2^S. We say that $\tau(s)$ is the *history* of s. A persistent function Δ is *ample* with respect to a history function τ if for every state s, either $\Delta(s) = enabled(s)$, or there is an action $\alpha \in \Delta(s)$ such that $succ(s, \alpha) \notin \tau(s)$. Thus, an ample function insists on exploring all enabled transitions or visiting some new state, which is not in the current history. If Δ is an ample function with respect to the history function τ, then $\Delta(s)$ is called an *ample set* at state s with respect to τ. Next, we define a history function τ_B and a numbering Ψ_B for breadth-first search so that Ψ_B is a witness for *any* ample function with respect to τ_B.

Partial-order reduction with breadth-first search The algorithm for constructing a partial-order reduction during BFS is a variation of the standard BFS algorithm. The idea is to use the current set of reached states as the history of a state. Further, since our aim is to perform the BFS symbolically, we simultaneously explore *all* states that are reachable in one step by the ample set of actions in the current frontier. More formally, let R_k be the set of reached states after the *k*th reachability step ($R_0 = \hat{S}$). Then define $\tau_B(s)$ as follows. If $s \in R_0$ then $\tau_B(s) = R_0$, else if $s \in R_{k+1} \backslash R_k$ then $\tau_B(s) = R_{k+1}$ (if $s \notin R_k$ for all k, the value of $\tau_B(s)$ is irrelevant). The pseudo code for the modified BFS algorithm is given in Figure 1.

Let G_B be the reduction of G with respect to the ample function chosen by the modified BFS. There are two types of states in the reduced state graph G_B—interior states that are never part of any search frontier, and frontier states that are part of some frontier exactly once. We define the mapping Ψ_B from the states of G_B to $\mathbb{N}$ as follows: (1) if s is never part of any frontier then $\Psi_B(s) = 1$, and (2) if s is part of the *k*th frontier and the total number of frontiers is N, then $\Psi_B(s) = N - k + 2$. Figure 2 illustrates the mapping Ψ_B for a reduced state graph with 4 frontiers. The shaded areas are the frontiers and the unshaded areas are the interior states. To prove that the modified BFS algorithm constructs a partial-order reduction G_B, it suffices to establish that the numbering Ψ_B is a witness for the ample function chosen by the algorithm.

Lemma 2. *Let Δ be an ample function with respect to the history function τ_B. If $\Delta(s) \neq enabled(s)$, then for some action $\alpha \in \Delta(s)$, we have $\Psi_B(succ(s, \alpha)) < \Psi_B(s)$.*

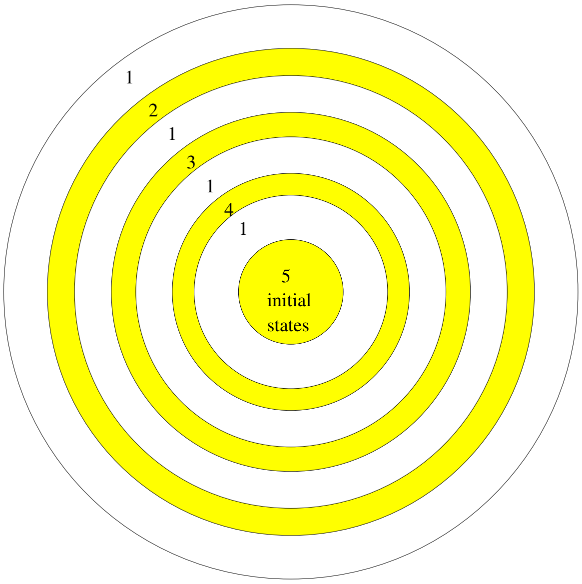

Fig. 2. Witness Ψ_B for states during BFS expansion

Theorem 3. *If G_B is the reduced graph obtained by the modified BFS algorithm from Figure 1, then G_B is a partial-order reduction of G.*

We note that depth-first search can also be used to construct a partial-order reduction. For this purpose, the set of states in the stack can be used as a history function [God96].

Verification using partial-order reductions Partial-order reductions are sound and complete for checking the invariance of local properties. We formalize local properties as independence-preserving regions. If there is an action sequence that originates at an initial state and leads to an independence-preserving region, then there exists some action sequence in any partial-order reduction that leads to the same region. A set $\sigma \subseteq S$ of states of a labeled transition graph G is an *independence-preserving region* if for all initial states s, whenever $succ(s, \overline{\alpha}) \in \sigma$ and $\overline{\beta}$ subsumes $\overline{\alpha}$, then there exists a prefix $\overline{\gamma}$ of $\overline{\beta}$ such that $succ(s, \overline{\gamma}) \in \sigma$. It follows that to verify that the given graph G satisfies an invariant σ (that is, no state outside σ is reachable in G), we can check whether some partial-order reduction of G satisfies the invariant σ, provided $S \backslash \sigma$ is an independence-preserving region of G.

3 Symbolic Computation of Ample Sets

Symbolic techniques capture transition sets and state sets of a labeled transition graph as boolean functions. The boolean functions are represented and manipulated as BDDs [BCMD92, McM93]. The functional representation of the transition relation is denoted by T, where $T(s, \alpha, t)$ is true iff $s \xrightarrow{\alpha} t$. Note that the BDD for T is built with variables that encode the present state, the next state, and the actions of G (these variables should not be confused with the variables of the labeled transition graph). In BFS, we denote the set of reached states by σ. First, we initialize σ to be the set of initial states. Second, the image σ_{new} of σ under T is computed:

```
σ_new := BDD representing Ŝ
repeat
    /* σ is the current set of reached states */
    σ := σ_new
    F := BDD representing the frontier of σ
    T^A := BDD representing the ample transition relation with respect to the history σ
    σ_next := image of F under T^A
    σ_new := σ ∨ σ_next
until σ = σ_new
```

Fig. 3. BDD-based modified breadth-first search algorithm

$\sigma_{new}(s) = \exists t.\exists\alpha.(\sigma(t) \wedge T(t, \alpha, s))$. The set of states that can be reached by the system in at most one step from σ is represented by $\sigma \vee \sigma_{new}$. The image computations are iterated until a fixpoint is reached. This fixpoint then represents the set of all reachable states. Figure 3 shows the modified BFS algorithm from Figure 1 (with partial-order reduction) rewritten with BDD operations. To implement this algorithm we need to produce an *ample transition relation* T^A at each reachability step.

A *persistent transition relation* T^P is a BDD encoding of a persistent function Δ such that $T^P(s, \alpha, t)$ iff $\alpha \in \Delta(s)$ and $s \xrightarrow{\alpha} t$. An ample transition relation T^A with respect to the history σ can be computed from T^P as

$$T^A(s, \alpha, t) \;=\; T^P(s, \alpha, t) \vee \big(\neg\exists\alpha'.\exists t'.(T^P(s, \alpha', t') \wedge \neg\sigma(t')) \wedge T(s, \alpha, t)\big).$$

It remains to be seen how we compute the persistent transition relation T^P. We now show that a heuristic for computing persistent sets in [God96] can be implemented symbolically to yield a persistent transition relation. We assume a model of the transition system annotated with processes and give a symbolic procedure for computing persistent sets in this model.

A *process structure* is a structured description of a labeled transition graph. Formally, it is a 4-tuple $\mathcal{P} = \langle \mathbb{P}, G, own, exec\rangle$ with the following components:

- A finite set $\mathbb{P} = \{p_1, p_2, \ldots, p_n\}$ of processes.
- A labeled transition graph $G = \langle V, S, \hat{S}, \Gamma, \rightarrow\rangle$.
- A function own from V to $\mathbb{P}$. We say that a variable v is *owned* by process $own(v)$.
- A function $exec$ from Γ to $\mathbb{P}$, with the following restriction: for all states s and t, all actions α, and all variables v, if $s \xrightarrow{\alpha} t$ and v changes value from s to t, then $own(v) = exec(\alpha)$. We say that action α is *executed* by process $exec(\alpha)$.

Observe that in this model, variables are read-shared, but write-exclusive. We write $act(p_i)$ to denote the set $\{\alpha \mid exec(\alpha) = p_i\}$ of actions that are executed by process p_i. Since each action is executed by exactly one process, act induces a partition on the set Γ of actions. We differentiate between *global state* (valuations for all variables) and *local state* of a process (valuations for the variables owned by the process). A global state s is an n-tuple $(s_1, s_2, \ldots, s_n)$ of local states s_i, one for each process $p_i \in \mathbb{P}$. We say that s_i is the local state of process p_i in s. An action $\alpha \in act(p_i)$ is *enabled in the local state* s_i of process p_i if there exists a global state s containing s_i such that α is enabled in s.

```
/* Let s = (s_1, s_2, ..., s_n) be the given global state */
P(s) := {p_k}, where p_k is some process enabled in s
repeat
  for every process p_i ∈ P(s) and every action α ∈ act(p_i) enabled in s_i do
    for every process p_j ∉ P(s) do
      if there exists an action β ∈ act(p_i) such that β is dependent with α then
        P(s) := P(s) ∪ {p_j}
      fi
    end
  end
until no more processes can be added to P(s)
```

Fig. 4. Algorithm for computing a persistent set $P(s)$ at state s

The algorithm for computing persistent sets is sketched in Figure 4. Given a state s, the algorithm finds a set of processes $P(s) \subseteq \mathbb{P}$ such that the set of actions executed by the processes in $P(s)$ is a persistent set at s. First, we initialize $P(s)$ with an arbitrary process p_k that executes an action enabled in s. Then we iteratively add more processes to $P(s)$, as follows, until no more processes can be added: for all processes $p_i \in P(s)$ and actions $\alpha \in act(p_i)$ that are enabled in the local state s_i of process p_i, we add process p_j to $P(s)$ if $act(p_j)$ executes an action β that is dependent with α. We call the resulting set $P(s)$ a *persistent set of processes* at s. The algorithm described above checks only a sufficient condition for persistence; it is possible to formulate more complex schemes that yield smaller persistent sets [God96].

Our aim is to compute a boolean function I such that $I(s,p)$ is true iff $p \in P(s)$ for the global state s. In effect, I encodes persistent sets of processes for all global states. For computing I, we make use of the following pre-computed relations:

- $enabledG(s, \alpha)$ is true iff action α is enabled in the global state s.
- $actionof(\alpha, p)$ is true iff $exec(\alpha) = p$.
- $enabledL(s, \alpha, p)$ is true iff $exec(\alpha) = p$ and α is enabled in the local state of p in s:

$$\begin{aligned} enabledL((s_1, s_2, s_3, ..., s_n), \alpha, p) \;=\; & \textstyle\bigvee_{1 \le i \le n}((p = p_i) \Rightarrow \\ & \exists s'_1 \ldots s'_{i-1} s'_{i+1} \ldots s'_n . \\ & enabledG((s'_1, s'_2, \ldots, s'_{i-1}, s_i, s'_{i+1}, \ldots, s'_n), \alpha)). \end{aligned}$$

- $dependent(\alpha, \beta)$ is true iff actions α and β are dependent. The dependence relation can be overapproximated by sufficient syntactic checks (e.g., disjointness of support variables) [God96].

For all i, $1 \le i \le n$, we compute $J_i(s,p)$ as follows:

$$J_i(s,p) \;=\; (p = p_i) \wedge \exists \alpha.(actionof(\alpha, p_i) \wedge enabledG(s, \alpha)).$$

The relation $J_i(s,p)$ is true iff $p = p_i$ and $act(p_i)$ contains at least action enabled in s. We then perform a fixpoint computation to compute the relation $I(s,p)$ as follows:

$$\begin{aligned} I_0(s,p) = \; & J_1(s,p) \vee \\ & (\neg \exists p . J_1(s,p) \wedge J_2(s,p)) \vee \\ & (\neg \exists p . (J_1(s,p) \vee J_2(s,p)) \wedge J_3(s,p)) \;\; \vee \ldots \\ & (\neg \exists p . (J_1(s,p) \vee J_2(s,p) \vee \ldots \vee J_{n-1}(s,p)) \wedge J_n(s,p)), \end{aligned}$$

$$I_{k+1}(s,p) = I_k(s,p) \vee$$
$$\exists p'.\exists\alpha.\exists\beta.(I_k(s,p') \wedge enabledL(s,\alpha,p') \wedge\ actionof(\alpha',p) \wedge dependent(\alpha,\beta)).$$

The relation $I_0(s,p)$ is true iff p is the process of least index that executes an action enabled in state s. The computation of I_{k+1} from I_k mimics the repeat-until loop of the algorithm in Figure 4. Then, the desired relation I is I_m for the smallest m such that $I_{m+1} = I_m$. Now the persistent transition relation T^P can be computed from I as follows:

$$T^P(s,\alpha,t) \ = \ \exists p.(I(s,p) \wedge actionof(\alpha,p)) \wedge T(s,\alpha,t).$$

4 Experiments

We consider two examples for experiments—a protocol for electing a leader among a set of processes and an asynchronous tree arbiter circuit. In both cases, we built Verilog models for the original system as well as the reduced system obtained by considering only those actions from a state that form an ample set. To impose interleaving semantics on a Verilog description, we added a scheduler to the model which nondeterministically schedules one process at a time. The symbolic reachability computation was performed with VIS [BHSV$^+$96]. The ample sets were manually computed as functions of global states and implemented using an "ample scheduler" that nondeterministically chooses one among a subset of enabled processes (corresponding to the ample set) at each state. While the algorithm given in Section 3 can be used to compute the ample sets automatically, the computations require the identification of actions in the representation of transition systems, which VIS does not support at present. The purpose of the experiments is to investigate if savings in BDD sizes could be obtained, if such ample sets were available, and the answer is in the affirmative.

4.1 Leader-election protocol

We consider the leader-election protocol described in [DKR82]. Extensive reduction in reachable state space sizes for this problem is reported by [HP94], using partial-order reductions with enumerative depth-first search of the state space.

There are N processes connected in a circular ring, with a finite FIFO unidirectional queue between adjacent processes. Each process has a unique id associated with it. A process receives messages from the queue immediately before it in the ring and can send messages to the queue immediately after it. Process p_i has a variable called $active_i$ associated with it. Initially, $active_i = 1$ for all processes. The objective of the protocol is to elect a unique leader process p_k such that $active_k = 1$. The messages put on the FIFO queues contain id's of processes currently active. Initially, each process sends its id to the next process and receives the id of the preceding process. Every process remembers the maximum id it has seen so far. As a result of comparing received process id's to the maximum id, a process may deactivate itself. Finally, only one process should remain active. To check the property that finally a unique leader exists using an independence-preserving invariant, we introduce a variable called $flag_i$ for each process p_i. As soon as process p_k knows that it is the leader, it asserts $flag_k$ and $flag_k$ does not change thereafter. The independence-preserving invariants to be checked are $\neg(flag_i \wedge flag_j)$, for all $i \neq j$.[4]

A process is enabled if there is at least one message in its input queue and at least one empty slot in its output queue. In every state, every enabled process has precisely one enabled

[4] This, of course, is not a complete specification of the protocol. A liveness property stating that eventually a process does become the leader is also needed.

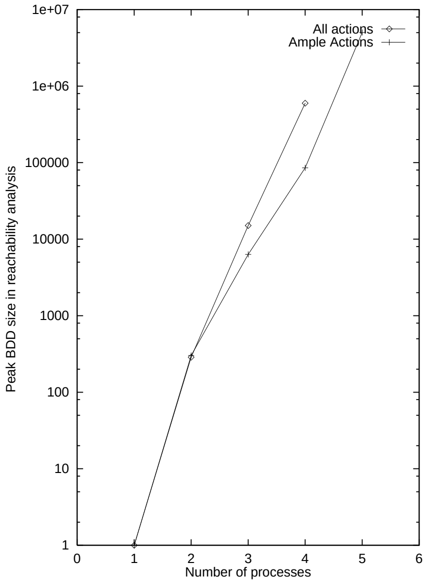

Fig. 5. Leader-election protocol: peak BDD sizes needed for reachability analysis

action. For each state, we consider the singleton set consisting of the action that is owned by the process with least index among the enabled processes. This set is a non-empty persistent set, because the protocol has the property that if multiple processes are enabled at the same time, the order in which the processes execute actions does not change the eventual outcome. The transition graph of the protocol is acyclic and hence every persistent set is ample.

Exponential partial-order reductions on this example are reported in [HP94] for a ring of 5 processes. There, however, the id's of the processes are assumed to be initialized to 0, 1, 2, 3, and 4 in circular order around the ring. Though the protocol is verified for this initial state, it is not verified for other initial states. For instance, if the id's are 0, 4, 1, 3, and 2, then the behavior of the protocol is very different. The former initialization requires a total of 15 messages passed around the ring before a leader is elected, whereas the latter requires 25 messages. In fact, there are certain lines of code in the protocol that are never exercised in the former case but are exercised in the latter. Our method verifies the correctness claim as stated in [DKR82], for *all* initial states.

Figure 5 shows the growth of peak BDD size during reachability analysis versus the number of processes. Two curves are shown—one for the entire transition relation and another for the ample transition relation. Note that reachability analysis with the entire transition relation runs out of memory for 5 processes. With partial-order reduction, in the case of 5 processes, VIS reports 3.76×10^{15} states visited. Thus, in this example, partial-order reduction coupled with symbolic search performs better than either method in isolation.

4.2 Asynchronous tree arbiter

The tree arbiter is an asynchronous circuit which solves the mutual-exclusion problem by building a tree of arbiter cells. The circuit does the arbitration in the form of an elimination tournament. An arbiter cell arbitrates between its two children. The leaves in the tree are processors that asynchronously demand access to a common resource. The n users at the

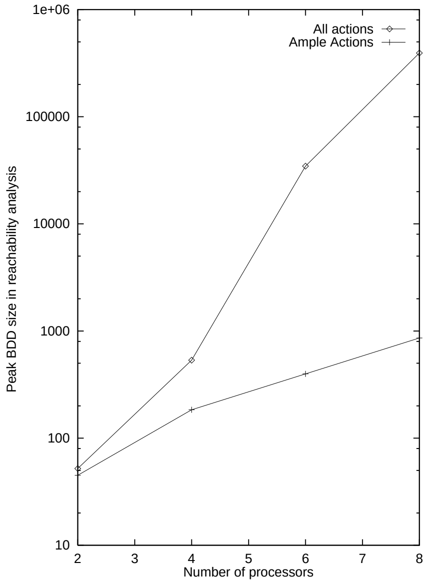

Fig. 6. Tree arbiter: peak BDD sizes needed for reachability analysis

lowest level in the tree are arbitrated by $\frac{n}{2}$ cells. The winners at this level are arbitrated at the next level and so on. If both children of an arbiter cell request, one of them is chosen nondeterministically. The requests propagate upward in the tree until the root cell is reached. It grants the resource to at most one of its children and the grant propagates downward to one of the processors. Similarly, when the resource is released by a processor, the request for release goes all the way up and is acknowledged all the way down. We model the arbiter cells as interacting asynchronously with each other. Four-phase signaling is used for communication between the arbiter cells. The circuit is verified in [Dil89].

We label all processes (processors and arbiter cells) by levels, starting at the leaves: at the lowest level, label the processors from left to right with the leftmost processor having the least label; then label the first level of arbiters from left to right; etc. Given such a labeling, it can be shown that at each state, the singleton set containing the enabled process with the least label is an ample set. Figure 6 shows the BDD sizes during reachability analysis, for the entire transition relation and the ample transition relation (8 processors induce a tree with three levels of arbitration). Note the log scale on the vertical axis of the plot. The memory requirement for the entire transition relation grows exponentially with the number of processors, whereas the memory for the ample transition relation grows polynomially. The difference in memory consumption also manifests itself in run time: for 8 processors, reachability analysis with the entire transition relation requires 1755 seconds, whereas the ample transition relation requires less than 1 second to complete.

4.3 Utilizing flexibility as "don't cares"

We implemented a new command in VIS called *ample_reach* that takes two transition relations (an ample transition relation and the entire transition relation) and does the following. At each step during reachability analysis, two images are computed, using the two transition relations. The image obtained by the ample transition relation is always a subset of the image obtained by the entire transition relation. Then, two new reached sets are computed, by taking the union of each of the two images with the reached set from the previous step. Finally, a heuristic BDD minimization function is called from the BDD package, to minimize the representation of a set that lies between the two reached sets. In both of the above examples, the heuristic minimization always returned the smaller reached set, that is, the one obtained from the ample transition relation. We believe that the reason for this is the exponential reduction in the number of states obtained using ample sets. However, if a system has asynchronous components operating with limited independence and there is limited partial-order reduction, this method might prove to be useful.

Acknowledgments Gerard Holzmann provided us with information on SPIN. Ken McMillan and Doron Peled contributed through discussions. The VIS group at UC Berkeley and Rajeev Ranjan in particular helped with the experiments.

References

[BCL91] J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic Model Checking with Partitioned Transition Relations. In *Proc. of the 28th Design Automation Conference*, pages 403–407, 1991.

[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.

[BHSV$^+$96] R.K. Brayton, G.D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R.K. Ranjan, S. Sarwary, T.R. Shiple, G. Swamy, and T. Villa. VIS: A System for Verification and Synthesis. In *Proc. of the 8th International Conference on Computer-Aided Verification*, vol. 1102 of *Lecture Notes in Computer Science*, pages 428–432. Springer, 1996.

[CP96] C.-T. Chou and D.A. Peled. Formal Verification of a Partial-Order Reduction Technique for Model Checking. In *Proc. of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, vol. 1055 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 1996.

[Dil89] D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. MIT Press, 1989.

[DKR82] D. Dolev, M. Klawe, and M. Rodeh. An $O(n \log n)$ Unidirectional Distributed Algorithm for Extrema Finding in a Circle. *Journal of Algorithms*, 3:245–260, 1982.

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

[GW94] P. Godefroid and P. Wolper. A Partial Approach to Model Checking. *Information and Computation*, 110:305–326, 1994.

[HP94] G.J. Holzmann and D.A. Peled. An Improvement in Formal Verification. In *Proc. of the 7th International Conference on Formal Description Techniques*, pages 197–211. Chapman & Hall, 1994.

[Maz88] A. Mazurkiewicz. Basic Notions of Trace Theory. In *Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, vol. 354 of *Lecture Notes in Computer Science*, pages 285–363. Springer, 1988.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[Pel93] D.A. Peled. All from One, One for All: On Model Checking Using Representatives. In *Proc. of the 5th International Conference on Computer-Aided Verification*, vol. 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993.

[RAP$^+$95] R.K. Ranjan, A. Aziz, B. Plessier, C. Pixley, and R.K. Brayton. Efficient Formal Design Verification: Data Structures + Algorithms. In *Workshop Notes of the International Workshop on Logic Synthesis*, 1995.

[Val91] A. Valmari. Stubborn Sets for Reduced State Space Generation. In *Advances in Petri Nets*, vol. 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer, 1991.