Combining Partial Order and Symmetry Reductions

E. Allen Emerson[1] * and Somesh Jha[2] ** and Doron Peled[3]

[1] Department of Computer Science, University of Texas, Austin, TX, USA
[2] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[3] Bell Laboratories, Lucent Technologies, 700 Mountain Ave., Murray Hill, NJ, USA

Abstract. Partial order based reduction techniques to reduce time and memory in model-checking procedures are becoming quite popular. Partial order reduction techniques exploit the independence of actions. Symmetry based reduction techniques exploit the inherent structure of the system to reduce the state space explored during model checking. We provide an abstract framework for combining partial-order and symmetry reductions. We also present algorithms which exploit both reduction techniques simultaneously.

1 Introduction

Partial order based methods exploit the independence of actions [6, 7, 11, 14, 15] to reduce the state space explosion in model-checking concurrent systems. The basic idea is that given a set of interleaving sequence of actions, one can define sequences that are equivalent up to reordering independent actions to be equivalent. As most specifications would not distinguish between equivalent sequences, one can consider a subset of sequences from each equivalence class. Thus, the reduction generates a state space that includes only a sufficiently big subset of of the sequences, representing all other equivalent ones. For example, assume that the actions α and β are independent. Independence means that it does not matter in what order the finite-state system executes the actions α and β. Thus, a sequence $u\alpha\beta v$ is equivalent to the sequence $u\beta\alpha v$. Therefore, if an algorithm considers the sequence $u\alpha\beta v$, it can omit the sequence $u\beta\alpha v$. Most methods work by exploring a subset of the actions enabled from a state. The subset is selected according to some constraints that guarantee that enough representatives, at least one from each equivalence class, will be generated.

* The research of the first author was supported by NSF under grant no. CCR-9415496 and Semiconductor Research Corporation under contract 96-DP-388.

** The reseach of the second author was sponsored in part by the National Science Foundation under grant no. CCR-8722633, by the Semiconductor Research Corporation under contract 92-DJ-294, and by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant F33615-93-1-1330.

Symmetry based methods exploit the architectural symmetry present in the system. For example, in a token ring composed of identical components, one can rotate the ring of processes without affecting the behavior of the system. Symmetry is present in any finite-state system composed of several identical components. Symmetry was first exploited in the reachability analysis of Petri Nets [8, 13]. Recently, symmetry based methods to avoid the state-explosion during model-checking were shown to be quite successful [3, 5, 9]. The basic idea is that the symmetry of the system induces an equivalence relation on the state space of the system. While performing model checking, one can discard the state s' if one has already explored an equivalent state s.

This paper combines the symmetry and partial order based reduction techniques. Since symmetry and partial order based methods explore different phenomena of the system, it is possible for both techniques to be applied simultaneously, obtaining better reduction than by applying each one of them separately. We show that this indeed is the case. In related work, Valmari has suggested to combine partial order reduction and symmetry for deadlock detection in colored Petri-Nets in [16].

The result shown here can also be interpreted more generally as combining two reduction techniques based on preserving equivalence (simulation) relations. In that respect, one can generalize the symmetry and the partial order reductions into other equivalence preserving reductions. Although preserving other reductions may involve a somewhat different, specialized proof, similar ideas of how to 'reconciliate' between equivalence relations in order to preserve the correctness of the checked property may be applied. Much of the effort was put to make the proofs of the combined reductions modular, separating as much as possible the arguments concerning the partial order reduction from the arguments concerning symmetry reduction.

The paper is organized as follows: Section 2 provides definitions used throughout the paper. Section 3 gives a framework for combining symmetry with independence, which will be used to explain and prove the combined reduction algorithms. Section 4 gives an algorithm which preserves LTL formulas without the nexttime operator. Section 5 provides an algorithm which preserves $CTL^\star$ without the nexttime operator. Section 7 concludes with some future directions and open problems. Due to limited space, the results of few lemmas are not provided here. They will appear in the full version of the paper.

2 Preliminaries

In this section we introduce various definitions used throughout the paper. Subsection 2.1 defines a labeled transition system. Temporal logics $CTL^\star$, $CTL^\star$-X, LTL, and LTL-X are not defined in this paper. The reader is referred to [2] for the syntax and semantics of these logics. Subsection 2.2 defines different preorders between LTSs. Subsection 3.2 defines what it means for two actions to be independent.

2.1 Labeled Transition System

Let AP be a set of atomic propositions. A *labeled transition system* (LTS) is 5-tuple $T = (S, R, L, Act, s_0)$, where

- S is a finite set of *states*,
- $R \subseteq S \times Act \times S$ is a *transition relation* ($(s, \alpha, s') \in R$ is also written as $s \xrightarrow{\alpha} s' \in R$).
- $L : S \rightarrow 2^{AP}$ is a *labeling function* which associates with each state a set of atomic propositions that are true in the state.
- Act is a finite set of actions.
- s_0 is the initial state.

The function $\alpha_T(s)$ returns the set α-successors of s in T. That is, $s' \in \alpha_T(s)$ iff $s \xrightarrow{\alpha} s' \in R$. An action α is said to be *enabled* from a state s in T if and only if there exists a state s' such that $s \xrightarrow{\alpha} s' \in R$. The symbol $en_T(s)$ denotes the set of actions enabled from the state s in T. An action α is called *invisible* in T iff for all s and s' such that $s \xrightarrow{\alpha} s' \in R$ we have that $L(s) = L(s')$. Basically, an invisible action does not change the truth of atomic propositions. The set of invisible actions in T is denoted by $\mathtt{invis}_T$. The set of visible actions is denoted by $\mathtt{vis}_T$.

2.2 Various pre-orders between processes

Given two LTSs $T_1 = (S_1, R_1, L_1, Act, s_{0,1})$ and $T_2 = (S_2, R_2, L_2, Act, s_{0,2})$, a relation $\mathcal{B} \subseteq S_1 \times S_2$ is called a *bisimulation* between T_1 and T_2 if and only if the following conditions hold:

- $s_{0,1} \; \mathcal{B} \; s_{0,2}$.
- Assume that $s \; \mathcal{B} \; s'$. Then the following conditions hold:
  - $L(s) = L(s')$
  - Given an arbitrary transition $s \xrightarrow{\alpha} s_1 \in R_1$, there exists $s_2 \in S_2$ such that $s' \xrightarrow{\alpha} s_2 \in R_2$ and $s_1 \; \mathcal{B} \; s_2$.
  - A symmetric condition holds with the roles of s' and s reversed.

T_1 and T_2 are said to be *bisimilar* (denoted by $T_1 \cong_B T_2$) if and only if there exists a bisimulation between T_1 and T_2.

Definition 1. Let $T_1 = (S_1, R_1, L_1, Act, s_{0,1})$ and $T_1 = (S_2, R_2, L_2, Act, s_{0,2})$ be two LTSs. Let $\mathcal{E} \subseteq S_1 \times S_2$ be a relation. Consider paths $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \cdots$ in T_1 and $\pi' = t_0 \xrightarrow{\beta_0} t_1 \xrightarrow{\beta_1} \cdots$ in T_2. Paths π and π' are called *stuttering $\mathcal{E}$-equivalent* if and only if there exists infinite sequences of natural numbers $i_0 = 0 < i_1 < i_2 < \cdots$ and $k_0 = 0 < k_1 < k_2 < \cdots$ such that for all $j \geq 0$ the following condition is true.

– For all $i_j \leq r < i_{j+1}$ and $k_j \leq m < k_{j+1}$, $s_r \ \mathcal{E} \ t_m$.

Paths π and π' are called *stuttering equivalent* if they are stuttering $\mathcal{L}$-equivalent where $s \ \mathcal{L} \ s'$ if and only if $L(s) = L(s')$. Sometimes, we will refer to the set of integers $\{i_j, i_j + 1, \cdots, i_{j+1} - 1\}$ and $\{k_j, k_j + 1, \cdots, k_{j+1} - 1\}$ as the *j-th blocks* B_j and B'_j.

Next, we define the notion of *stuttering bisimulation*. Stuttering bisimulation is similar to bisimulation, but each LTS is allowed to take several steps to simulate a path of the other LTS. Given two LTSs $T_1 = (S_1, R_1, L_1, Act_1, s_{0,1})$ and $T_2 = (S_2, R_2, L_2, Act_2, s_{0,2})$, a relation $\mathcal{E} \subseteq S_1 \times S_2$ is called a *stuttering bisimulation* between T_1 and T_2 if and only if the following conditions hold:

– $s_{0,1} \ \mathcal{E} \ s_{0,2}$.
– If $s \ \mathcal{E} \ s'$, then the following conditions hold:
  - $L(s) = L(s')$
  - For every path π starting from s in T_1 there exists a stuttering $\mathcal{E}$-equivalent path π' starting from s' in T_2. See definition 1 for the explanation of stuttering $\mathcal{E}$-equivalent.
  - The same condition as the previous one holds but with the roles of s and s' reversed.

T_1 and T_2 are said to be *stuttering bisimilar* if and only if there exists a stuttering bisimulation between them. We denote this by $T_1 \cong_{SB} T_2$.

Notice that a stuttering bisimulation relation cannot distinguish between next states. Therefore, it is not surprising that stuttering bisimulation preserves the truth of CTL^* formula without the next time operator. The proof of this theorem first appeared in [1].

Theorem 2. Let f be a formula in $CTL^\star$-X. Let T_1 and T_2 be two stuttering bisimilar LTSs. Let $\mathcal{E}$ be a stuttering bisimulation relation between T_1 and T_2. If $s \ \mathcal{E} \ s'$, then $T_1, s \models f$ if and only if $T_2, s' \models f$.

$T_1 = (S_1, R_1, L_1, Act, s_{0,1})$ and $T_2 = (S_2, R_2, L_2, Act, s_{0,2})$ are said to be *stuttering path equivalent* (denoted by $T_1 \cong_{SPE} T_2$) if and only if

– For every path π starting from $s_{0,1}$ in T_1 there exists a stuttering equivalent path π' starting from $s_{0,2}$ in T_2.
– A symmetric condition holds with the roles of $s_{0,1}$ and $s_{0,2}$ reversed.

Theorem 3. Let f be a formula in LTL-X. Let $T_1 = (S_1, R_1, L_1, Act, s_{0,1})$ and $T_2 = (S_2, R_2, L_2, Act, s_{0,2})$ be two stuttering path equivalent LTSs.

$$s_{0,1} \models f \Leftrightarrow s_{0,2} \models f$$

Lemma 4. $T_1 \cong_B T_2$ implies that $T_1 \cong_{SB} T_2$. Similarly, $T_1 \cong_{SB} T_2$ implies that $T_1 \cong_{SPE} T_2$.

Lemma 5. The pre-orders $\cong_B$, $\cong_{SB}$ and $\cong_{SPE}$ are transitive.

3 Reconciling Symmetry and Commutativity

3.1 Selection Function and Abstract Structure

Definition 6. Let $T = (S, R, L, Act, s_0)$ be an LTS. A function $h : S \to S$ is a *selection function* if and only if there exists a bisimulation relation $\mathcal{B} \subseteq S \times S$ between T and T such that;

- For all $s \in S$, $s\ \mathcal{B}\ h(s)$
- $s\ \mathcal{B}\ s'$ implies that $h(s) = h(s')$

We say that h *preserves* the bisimulation relation $\mathcal{B}$.

Intuitively, the function h picks a representative from each equivalence class of S induced by the bisimulation $\mathcal{B}$. Given a selection function h, a reduced state space can consist of the representative states (i.e., those in the range of h) with edges between them:

Definition 7. Given a selection function h on an LTS $T = (S, R, L, Act, s_0)$, define the corresponding *abstract LTS* $T_h = (S_h, R_h, L_h, Act, h(s_0))$ in the following manner:

- $S_h = h(S)$.
- $r_1 \xrightarrow{\alpha} r_2 \in R_h$ if and only if there exists $s \in S$ such that $r_1 \xrightarrow{\alpha} s \in R$ and $h(s) = r_2$.
- For all $r \in S_h$, $L_h(r) = L(r)$.

We will show that the abstract (reduced) state space is bisimilar to the original state space. Thus, according to Theorem 2, preserves all the $CTL^\star$-X properties.

Lemma 8. Given an LTS $T = (S, R, L, Act, s_0)$ and a selection function h, T and T_h are bisimilar.

Proof: Let $\mathcal{B} \subseteq S \times S$ be a bisimulation such that h is a selection function. Construct $\mathcal{B}_h \subseteq S \times S_h$ in the following manner:

$$s\ \mathcal{B}_h\ r \Leftrightarrow s\ \mathcal{B}\ r$$

We will prove that B_h is a bisimulation relation. Assume that $s\ \mathcal{B}_h\ r$. It is obvious that the labels of r and s match.

- Assume that $s \xrightarrow{\alpha} s' \in R$. We need to show that $r \xrightarrow{\alpha} r' \in R_h$, with $s'\ \mathcal{B}_h\ r'$. Since $s\ \mathcal{B}\ r$, there exists $r_1 \in S$ such that $r \xrightarrow{\alpha} r_1 \in R$ and $s'\ \mathcal{B}\ r_1$. Let $r' = h(r_1)$. By definition, $r \xrightarrow{\alpha} r' \in R_h$ and $r_1\ \mathcal{B}\ r'$. By transitivity, $s'\ \mathcal{B}\ r'$, which implies that $s'\ \mathcal{B}_h\ r'$.
- Assume that $r \xrightarrow{\alpha} r' \in R_h$. We need to show that $s \xrightarrow{\alpha} s'$ such that $s'\ \mathcal{B}_h\ r'$. The definition of R_h implies that there exists r_1 such that $r \xrightarrow{\alpha} r_1 \in R$ and $h(r_1) = r'$. Since $s\ \mathcal{B}\ r$, there exists s' such that $s \xrightarrow{\alpha} s'$ and $s'\ \mathcal{B}\ r_1$. Using the fact that $r_1\ \mathcal{B}\ r'$ (recall that $h(r_1) = r'$ and h is bisimulation preserving) and transitivity of $\mathcal{B}$, $s'\ \mathcal{B}\ r'$. This implies that $s'\ \mathcal{B}_h\ r'$. ∎

3.2 Independent Actions

Now we define the concept of independent actions.

Definition 9. Let $T = (S, R, L, Act, s_0)$ be an LTS. An *independence relation* on actions is an irreflexive and symmetric relation $I \subseteq Act \times Act$ such that each pair of actions $(\alpha, \beta) \in I$ (called *independent actions*) it must hold that for each $s \in S$

- If $\{\alpha, \beta\} \subseteq en_T(s)$, then for each state $s' \in \alpha_T(s)$ we have that $\beta \in en_T(s')$.
- If $\{\alpha, \beta\} \subseteq en_T(s)$, then there exists a path from $s \xrightarrow{\alpha} s_1 \xrightarrow{\beta} s'$ in T iff there exists a path $s \xrightarrow{\beta} r_1 \xrightarrow{\alpha} s'$ in T.

The first condition states that if α and β are independent, then executing α from a state s, does not disable the action β. The second condition states that independent actions are commutative. Notice that the relation I is an independence relation with respect to a particular LTS T.

The lemma given below states that if I is an independence relation for T, then I is also an independence relation for T_h. This means that given an independence relation for T, we can use the same independence relation while performing partial-order reduction on T_h.

Lemma 10. Let I be an independence relation for a LTS $T = (S, R, L, Act, s_0)$ and h a bisimulation preserving selection function. In this case, I is also an independence relation for the corresponding abstract LTS T_h (denoted by $(S_h, R_h, L_h, Act, h(s_0)))$.

Proof: Let I be an independence relation for T. We will prove that I is also an independence relation for T_h. Let $\mathcal{B}$ be a bisimulation relation between T and T which is preserved by h. Assume that $(\alpha, \beta) \in I$. Corresponding to the two conditions in the definition of the independence relation we have the following two cases:

- Assume that $\{\alpha, \beta\} \subseteq en_{T_h}(r)$. Let $r' \in \alpha_{T_h}(r)$. We have to prove that $\beta \in en_{T_h}(r')$. By definition, there exists $s \in S$ such that $r \xrightarrow{\alpha} s$ and $h(s) = r'$. Since I is an independence relation for T, $\beta \in en_T(s)$. Since h is bisimulation preserving, we also have that $s\ \mathcal{B}\ r'$. Therefore, $\beta \in en_T(r')$, which in turn implies that $\beta \in en_{T_h}(r')$.
- Assume that $\{\alpha, \beta\} \subseteq en_{T_h}(r)$. Now suppose that there exists a path $r \xrightarrow{\alpha} r_1 \xrightarrow{\beta} r'$ in T_h. Let $s \in S$ such that $r \xrightarrow{\alpha} s \in R$ and $h(s) = r_1$. Also assume that $r_1 \xrightarrow{\beta} t_1 \in R$ and $h(t_1) = r'$. Since $\beta \in en_T(s)$, we can construct a path $r \xrightarrow{\alpha} s \xrightarrow{\beta} s'$ in T such that $s'\ \mathcal{B}\ t_1$. Since I is an independence relation for T, there exists a path $r \xrightarrow{\beta} s_1 \xrightarrow{\alpha} s'$ in T. A transition $h(s_1) \xrightarrow{\alpha} s''$ such that $s'\ \mathcal{B}\ s''$ exists because $s_1\ \mathcal{B}\ h(s_1)$. By transitivity, $s''\ \mathcal{B}\ t_1$. Therefore, $h(s'') = h(t_1) = r'$. Hence, $r \xrightarrow{\beta} h(s_1) \xrightarrow{\alpha} r'$ is a path in T_h.

Only the second part of the proof uses the fact that $s \mathcal{B} s'$ implies that $h(s) = h(s')$. ∎

3.3 Symmetry

Let Perm(S) be the group of permutations of the finite set S. When we say G acts on S, we mean that G is a subgroup of Perm(S). If G is a subgroup of H, we denote it by $G \leq H$. Next, we define the concept of a symmetry group G.

Definition 11. Given a LTS $T = (S, R, L, Act, s_0)$, a group G acting on S is called a *symmetry group* of T iff

- For all $\alpha \in Act$ and for all $\sigma \in G$, $s \xrightarrow{\alpha} s'$ iff $\sigma(s) \xrightarrow{\alpha} \sigma(s')$.
- For all $\sigma \in G$, $L(s) = L(\sigma(s))$.

Notice that if we are interested in checking a temporal formula f, the labeling function of the LTS can be restricted to the atomic propositions occurring in f. Therefore, all the restrictions on labelings given above have to only hold for the atomic propositions occurring in the temporal formula f of interest. We say that s and s' are in the same *orbit* iff there exists a $\sigma \in G$ such that $\sigma(s) = s'$. $\Theta \subseteq S \times S$ is the *orbit relation* induced by the symmetry group G. Given a LTS $T = (S, R, L, Act, s_0)$ and a symmetry group G acting on S, we define a *representative function* $\xi : S \to S$. The function ξ has two properties:

- s and $\xi(s)$ are in the same orbit.
- If s and s' are in the same orbit, then $\xi(s) = \xi(s')$.

The function ξ maps a state to an unique representative in its orbit. The lemma given below states that ξ is a bisimulation preserving selection function for T. This means that our entire framework automatically gives a method for combining partial-order and symmetry reductions. It is easy to show the following:

Lemma 12. Assume that we are given a LTS $T = (S, R, L, Act, s_0)$ and a symmetry group G acting on S. Let ξ be a representation function corresponding to G. In this case, ξ is a bisimulation preserving selection function.

In the definition of the symmetry group given at the beginning of this subsection we did not allow the actions to be permuted. This might seem overly restrictive. Now we will allow the symmetry group to permute states and actions simultaneously. Next, we will prove that this new seemingly more powerful notion of symmetry is equivalent to the definition of symmetry given before. Assume that we are given an LTS T and a symmetry group G according to the definition 13. We construct an LTS T_1 from T by relabeling actions such that G is a symmetry group for T_1 using definition 11. The group Perm(S) $\times$ Perm(Act) is the group of all permutations (σ_1, σ_2) such that $\sigma_1 \in$ Perm(S) and $\sigma_2 \in$ Perm(Act). Given a permutation $\psi = (\sigma_1, \sigma_2) \in$ Perm(S) $\times$ Perm(Act), for all $s \in S$ and $\alpha \in Act$ we define $\psi(s) = \sigma_1(s)$ and $\psi(\alpha) = \sigma_2(\alpha)$.

Definition 13. Given a LTS $T = (S, R, L, Act, s_0)$, a group $G \leq \mathrm{Perm}(S) \times \mathrm{Perm}(Act)$ is called a *symmetry group* of T iff

- For $\psi \in G$, $s \xrightarrow{\alpha} s'$ iff $\psi(s) \xrightarrow{\psi(\alpha)} \psi(s')$.
- For all $\sigma \in G$, $L(s) = L(\sigma(s))$.

The *orbit* of an action $\alpha \in Act$ (denoted by $\theta_G(\alpha)$) is the following set:

$$\theta_G(\alpha) = \{\beta \mid \exists \psi \in G\ (\psi(\alpha) = \beta)\}$$

Let I be an independence relation on the LTS T. Let G be the symmetry group of T according to definition 13. Define $\Theta_G(I) \subseteq Act \times Act$ in the following manner:

- $(\alpha, \beta) \in \Theta_G(I)$ if and only if there exists a $\psi \in G$ such that $(\psi(\alpha), \psi(\beta)) \in I$.

The lemma given below states that if I is an independence relation for T, then $\Theta_G(I)$ is also an independence relation for T.

Lemma 14. Let $T = (S, R, L, Act, s_0)$ be a LTS and $G \leq \mathrm{Perm}(S) \times \mathrm{Perm}(Act)$ be a symmetry group of T. If I is an independence relation on T, then $\Theta_G(I)$ is an independence relation on T.

Proof of the lemma is omitted because of space restrictions. Now we can assume that we are working with $\Theta_G(I)$ instead of I. Notice that in general, $\Theta_G(I)$ can be much larger than I. The lemma given below states that the property of an action being invisible is an invariant for an orbit.

Lemma 15. Let $\sigma \in G$ be an arbitrary permutation in G. An action $\alpha \in \mathtt{invis}_T$ iff $\sigma(\alpha) \in \mathtt{invis}_T$.

$\Theta_G(Act)$ denotes the set of orbits of the actions. Given an LTS T (denoted by (S, R, L, Act, s_0)), a symmetry group $G \leq \mathrm{Perm}(S) \times \mathrm{Perm}(Act)$ (according to the definition 13) and an independence relation $I \subseteq Act \times Act$, we construct a symmetry group $G_1 \leq \mathrm{Perm}(S)$, an LTS T_1 (denoted by $(S_1, R_1, L_1, \Theta_G(Act), s_0)$), and an independence relation $I_1 \subseteq \Theta_G(Act) \times \Theta_G(Act)$ in the following manner:

- $S_1 = S$.
- $L_1(s) = L(s)$.
- $s \xrightarrow{\theta_G(\alpha)} s' \in R_1$ iff $s \xrightarrow{\alpha} s' \in R$.
- $\sigma_1 \in G_1$ iff there exists σ_2 such that $(\sigma_1, \sigma_2) \in G$.
- $\theta_G(\alpha)\ I_1\ \theta_G(\beta)$ iff for all $\alpha' \in \theta_G(\alpha)$ and for all $\beta' \in \theta_G(\beta)$ we have that $\alpha'\ I\ \beta'$.

The lemma given below states that definition 11 can be used without loss of generality.

Lemma 16. Let G be a symmetry group of an LTS T using definition 13. Let T_1 and G_1 be constructed as before. In this case G_1 is the symmetry group of T_1 according to the definition 11

Proof: Immediate from the construction of T_1. ∎

4 Algorithm for preserving *LTL-X*

Let $T = (S, R, L, Act, s_0)$ be an LTS and h be a bisimulation preserving selection function. In this section we will provide an algorithm which performs partial-order reduction and the reduction corresponding to the selection function h simultaneously. Basically, we describe an algorithm which performs the partial-order reduction on the abstract LTS $T_h = (S_h, R_h, L_h, Act, h(s_0))$, but does not require the explicit construction of T_h. First we present an algorithm which uses the structure T_h. This algorithm is only given for the sake of the proof.

```
1  push(h(s_0))
2  expand-node(h(s_0))

3  function expand-node(s)
4  working-set(s) = ample(s)
5  while working-set(s) ≠ φ do
6  α = some action in ample(s)
7  working-set = working-set(s)\{α}
8  for all s' ∈ α_{T_h}(s) do
9  if (new(s')) then
10 push(s')
11 expand-node(s')
12 create-edge(s,α,s')
13 fi
14 end for all
15 end while
16 mark s as explored.
17 end expand-node
```

Fig. 1. State space expansion algorithm **(A1)**

The routine new(s) checks that the state s has not been explored. The function push(s) pushes the state s onto the search stack. We also assume that when a state s is marked explored (line 16), it is popped from the search stack. $\mathcal{R}$ is called a *run* of the algorithm **A1** if and only if $\mathcal{R}$ is an execution of the algorithm **A1** where the sets $\texttt{ample}(s) \subseteq en_{T_h}(s)$ are chosen according to the rules **C1-h**, **C2-h** and **C3-h**

- **(C1-h)** For no action $\alpha \in Act \backslash \texttt{ample}(s)$ that is dependent on some action in $\texttt{ample}(s)$ there exists a path π in T_h (starting from s) such that α appears in π before an action from $\texttt{ample}(s)$ appears on π.
- **(C2-h)** If $\texttt{ample}(s)$ is a proper subset of the actions enabled from s in T_h, then for no action $\alpha \in \texttt{ample}(s)$ it holds that a state in the set $\alpha_{T_h}(s)$ is on

the search stack.

- **(C3-h)** If $\texttt{ample}(s)$ is a *proper* subset of $en_{T_h}(s)$, then none of the actions in $\texttt{ample}(s)$ are visible in T_h.

The following theorem states that any run of the algorithm **A1** produces a structure which is stuttering path equivalent to T_h.

Theorem 17. Let T' be the LTS produced by an arbitrary run of the algorithm **A1**. In this case $T_h \cong_{SPE} T'$.

Proof: See [11]. ∎

Now we modify the algorithm **A1** to produce algorithm **A2**. Algorithm **A2** works on the LTS T, but because of some modifications it behaves as if it is performing the partial order reduction on the LTS T_h. Algorithm **A2** is constructed from **A1** by changing lines 8, 9, 10, 11, and 12. We reproduce the whole algorithm for convenience, but mark the changed lines with a (**). $\mathcal{R}$ is called a *run* of the

1 push($h(s_0)$)
2 expand-node($h(s_0)$)

3 **function** expand-node(s)
4 working-set(s) = $\texttt{ample}(s)$
5 **while** working-set(s) $\neq \phi$ **do**
6 α = some action in $\texttt{ample}(s)$
7 working-set = working-set(s)$\backslash\{\alpha\}$
8 **for all** $s' \in \alpha_T(s)$ **do** (**)
9 **if** (new($h(s')$)) **then** (**)
10 push($h(s')$) (**)
11 expand-node($h(s')$) (**)
12 create-edge(s,α,$h(s')$) (**)
13 **fi**
14 **end for all**
15 **end while**
16 mark s as explored.
17 end expand-node

Fig. 2. State space expansion algorithm **(A2)**

algorithm **A2** if and only if $\mathcal{R}$ is an execution of the algorithm **A2** where the sets $\texttt{ample}(s) \subseteq en_T(s)$ are chosen according to the rules **C1**, **C2**, and **C3**.

- **(C1)** For no action $\alpha \in Act\backslash\texttt{ample}(s)$ that is dependent on some action in $\texttt{ample}(s)$ there exists a path π in T (starting from s) such that α appears in π before an action from $\texttt{ample}(s)$ appears on π.

- **(C2)** If $\texttt{ample}(s)$ is a proper subset of the actions enabled from s in T, then for no action $\alpha \in \texttt{ample}(s)$ it holds that a state in the set $h(\alpha_T(s)))$ is on the search stack.
- **(C3)** If $\texttt{ample}(s)$ is a *proper* subset of $en_T(s)$, then none of the actions in $\texttt{ample}(s)$ are visible in T.

The lemma given below will be used in our main theorem.

Lemma 18. Let $T = (S, R, L, Act, s_0)$ be an LTS and h a bisimulation preserving selection function. Let T_h be the corresponding abstract LTS. Then, we have the following conditions:

- Let $s \in S_h$. There exists a path $s = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \cdots$ in T if and only if there exists a path $s = t_0 \xrightarrow{\alpha_0} t_1 \xrightarrow{\alpha_1} \cdots$ in T_h. Notice that exactly the same actions appear in the two paths.
- An action α is visible in T if and only if it is visible in T_h.
- For all $s \in S_h$, $en_T(s) \;=\; en_{T_h}(s)$.

Proof: The results are a direct consequence of the fact that T and T_h are bisimilar (see lemma 8). ∎

Next, we prove that given a run of the algorithm **A2** there exists a run of the algorithm **A1** such that both runs produce the same LTS. We must emphasize again that algorithm **A1** only exists for the sake of the proof. In practice, **A2** will be implemented. The basic idea of the theorem is to run **A1** and **A2** in lockstep and show that the ample sets which satisfy conditions **C1**, **C2**, and **C3** for algorithm **A2** also satisfy conditions **C1-h**, **C2-h**, and **C3-h** for algorithm **A1** at each step.

Theorem 19. For every run $\mathcal{R}$ of the algorithm **A2** there exists a run $\mathcal{R}'$ of the algorithm **A1** such that the LTS produced by the two runs are the same.

Proof: We will construct a run $\mathcal{R}'$ of the algorithm **A1** as we trace the execution corresponding to the run $\mathcal{R}$ of the algorithm **A2**. At each point we will prove that the following invariants hold:

- If the run $\mathcal{R}$ chooses a set $\texttt{ample}(s)$ in line 4 which satisfies conditions **C1**, **C2**, and **C3**, then $\texttt{ample}(s)$ satisfies, **C1-h**, **C2-h** and **C3-h** for the run $\mathcal{R}'$.
- The state of the two runs are the same, i.e., the stacks have the same states and the same states are marked explored.

Initially, the invariants hold because both the runs push $h(s_0)$ on the stack. Lets say at some point in the execution the run $\mathcal{R}$ of the algorithm **A2** chooses a set $\texttt{ample}(s)$ on line 4 which satisfies conditions **C1**, **C2**, and **C3**. Because of lemma 18, $\texttt{ample}(s)$ also satisfies conditions **C1-h** and **C3-h**. Consider a state $r \in \alpha_{T_h}(s)$ where $\alpha \in \texttt{ample}(s)$. Notice that by definition there exists a state $s' \in \alpha_T(s)$ such that $r = h(s')$. Now it is obvious from condition **C2** that r

cannot be on the search stack. So $\texttt{ample}(s)$ considered by the run $\mathcal{R}$ satisfies condition **C2-h** for the run $\mathcal{R}'$. Also notice that lemma 18 implies that

$$(\texttt{ample}(s) \neq en_T(s)) \Leftrightarrow (\texttt{ample}(s) \neq en_{T_h}(s))$$

Now we advance the two runs, and assume that they consider the states in the same order in the **for all** loop starting at line 8. ∎

The theorem given below states that any run of the algorithm **A2** produces a LTS which is stuttering path equivalent to T.

Theorem 20. Let $\mathcal{R}$ be an arbitrary run of the algorithm **A2**. Let T' be the LTS produced by the run $\mathcal{R}$. Then we have that $T \cong_{SPE} T'$.

Proof: Let $\mathcal{R}'$ be the run of the algorithm **A1** which produces the same LTS as the run $\mathcal{R}$. Run $\mathcal{R}'$ exists because of theorem 19. By theorem 17 $T' \cong_{SPE} T_h$. Now lemmas 4 and 5 imply that $T \cong_{SPE} T'$. ∎

Notice that because of theorem 3 T and T' satisfy the same LTL-X formulas. Therefore, one can check a specification given in LTL-X on the smaller LTS.

5 Algorithm Preserving $CTL^{\star}$-X

The algorithm given in the previous section only preserved the existence of equivalent paths from the initial state. The semantics of branching time logics (like $CTL^{\star}$) are based on computation trees. Therefore, these logics can distinguish the branching structure of a node. Hence, to preserve branching time logics one has to put more stringent restrictions on the set $\texttt{ample}(s)$ considered by the algorithms. We call $\mathcal{R}$ a run of the of the algorithm **A1** if the $\texttt{ample}(s)$ satisfies the following condition in addition to conditions **C1-h**, **C2-h**, and **C3-h**.

- **(C4-h)** The set $\texttt{ample}(s)$ is a singleton set or $\texttt{ample}(s) = en_{T_h}(s)$.

In a similar manner, We call $\mathcal{R}$ a run of the of the algorithm **A2** if the $\texttt{ample}(s)$ satisfies the following condition in addition to conditions **C1**, **C2**, and **C3**.

- **(C4)** The set $\texttt{ample}(s)$ is a singleton set or $\texttt{ample}(s) = en_T(s)$.

The treatment is exactly the same as in section 4. Therefore, we will skip all the proofs. The proofs will use lemma 18 to establish that condition **C4** implies condition **C4-h**. An LTS $T = (S, R, L, Act, s_0)$ is called *deterministic* if and only if for all $s \in S$ and $\alpha \in Act$ we have that $|\alpha(s)| \leq 1$. We assume that the LTS T in question is deterministic. Notice that given a deterministic LTS T and a slection function h, T_h is a detereministic LTS. This restriction is needed for the proof of the following theorem.

Theorem 21. Let T' be the LTS produced by an arbitrary run of the algorithm **A1**. In this case $T_h \cong_{SB} T'$.

Proof: See [6, 12]. ∎

The proof of theorem given below is exactly the same as the proof of the theorem 19.

Theorem 22. For every run $\mathcal{R}$ of the algorithm **A2** there exists a run $\mathcal{R}'$ of the algorithm **A1** such that the LTS produced by the two runs are the same.

The theorem given below states that any run of the algorithm **A2** produces a LTS which is stuttering bisimilar to T.

Theorem 23. Let $\mathcal{R}$ be an arbitrary run of the algorithm **A2**. Let T' be the LTS produced by the run $\mathcal{R}$. Then we have that $T \cong_{SB} T'$.

Proof: Let $\mathcal{R}'$ be the run of the algorithm **A1** which produces the same LTS as the run $\mathcal{R}$. Run $\mathcal{R}'$ exists because of theorem 22. By theorem 21 $T' \cong_{SB} T_h$. By lemmas 4 and 5 $T \cong_{SB} T'$. ∎

Notice that because of theorem 2 T and T' satisfy the same $CTL^{\star}$-X formulas. Therefore, one can check a specification given in $CTL^{\star}$-X on the smaller LTS T' instead of T.

6 Example

In this section we given an example to illustrate our ideas. Figure 3 shows a solution to the two process mutual exclusion problem. N_i denotes that process i is the *neutral* section. T_i is the *trying* region for the process i. C_i signals that process i is in the *critical* section. Since we are only dealing with two processes, $i = 1$ or $i = 2$. Whenever process 2 makes a transition from N_2 to T_2, it sets an auxiliary variable $t = 1$. This signals the fact that process 1 can move into its critical section. A symmetric transition appears in process 1. It is obvious that exchanging indices 1 and 2 is a symmetry for this system. Let G be the corresponding symmetry group. There are 8 possible actions corresponding to the transitions. These are shown below:

χ_1	$N_1 \rightarrow T_1$	χ_2	$N_2 \rightarrow T_2$
α_1	$T_1 \rightarrow C_1$	α_2	$T_2 \rightarrow C_2$
β_1	$N_1 \rightarrow T_1$	β_2	$N_2 \rightarrow T_2$
δ_1	$C_1 \rightarrow N_1$	δ_2	$C_2 \rightarrow N_2$

Following the discussion in subsection 3.3, actions with the same name but different indices are in the same orbit under the action of the group G. For example, α_1 and α_2 are in the same orbit. Renaming the actions and performing the symmetry reductions we get the abstract structure given in the Figure 4. From the figure it should be clear what the representative function is. Also, notice that action α and β are independent. Now performing partial-order reduction on the abstract structure we get the structure given in Figure 5.

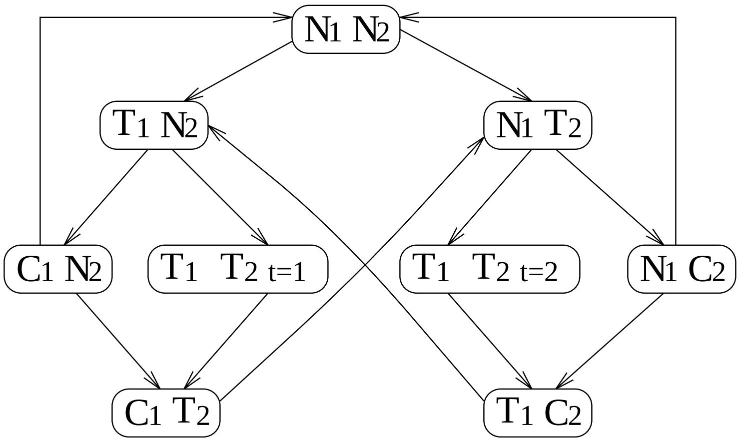

Fig. 3. Token Ring

7 Conclusion

This paper describes techniques to combine partial-order and symmetry reduction methods. In the future, we would like to implement our methods on some existing verification tools and try some examples. Other interesting problem is to derive symmetry and independence information from the description of the LTS being verified. Presently, most verification systems rely on the user to provide this information. We would also like to investigate whether some other reduction techniques could be combined using similar ideas.

References

1. M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.
2. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.
3. E.M. Clarke, T.Filkorn, and S.Jha. Exploiting symmetry in temporal logic model checking. In Courcoubetis [4].
4. C. Courcoubetis, editor. *Proceedings of the Fifth Workshop on Computer-Aided Verification*, volume 697 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1993.
5. E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. In Courcoubetis [4].

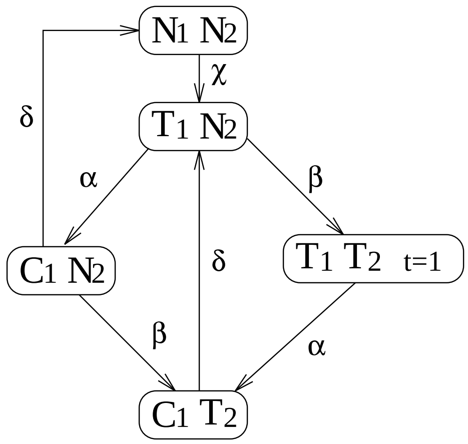

Fig. 4. Quotient Structure

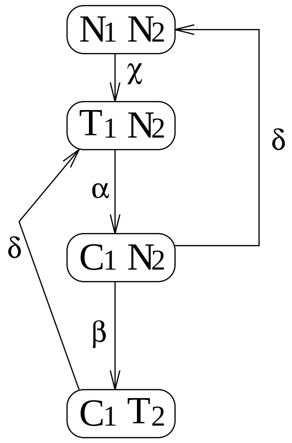

Fig. 5. Quotient Structure with PO reduction

6. Rob Gerth, Ruurd Kuiper, Doron Peled, and Wojciech Penczek. A partial order approach to branching time logic model checking. In *Third Israel Symposium on Theory on Computing and Systems*, pages 130–139, Tel Aviv, Israel, 1995. IEEE.
7. P. Godefroid. Using partial orders to improve automatic verification methods. In Kurshan and Clarke [10].
8. P. Huber, A.M. Jensen, L.O. Jepsen, and K. Jensen. Towards reachability trees for high-level petri nets. *In G. Rozenberg, editor, Advances on Petri Nets*, pages 215–233, 1984.
9. C.W. Ip and D. Dill. Better verification through symmetry. In L. Claesen, editor, *Proceedings of the Eleventh International Symposium on Computer Hardware Description Languages and their Applications*. North-Holland, April 1993.

10. R. P. Kurshan and E. M. Clarke, editors. *Proceedings of the 1990 Workshop on Computer-Aided Verification*. Springer-Verlag, June 1990.

11. Doron Peled. All from one, one from all: on model checking using representatives. In *5th International Conference on Computer Aided Verification, Greece*, number 697 in LNCS, pages 409–423, Elounda Crete, Greece, 1993. Springer-Verlag.

12. Doron Peled. Partial order reduction: Linear and branching temporal logics and process algebras. In *Partial Orders Methods in Verification*, DIMACS, Princeton, NJ, USA, 1996. American Mathematical Society.

13. P.H. Starke. Reachability analysis of petri nets using symmetries. *Syst. Anal. Model. Simul.*, 8(4/5):293–303, 1991.

14. A. Valmari. Stubborn sets for reduced state space generation. In *Proceedings of the Tenth International Conference on Application and Theory of Petri Nets*, 1989.

15. A. Valmari. A stubborn attack on the state explosion problem. In Kurshan and Clarke [10].

16. A. Valmari. Stubborn sets of colored petri nets. In *Proceedings of the 12th International Conference on Application and Theory of Petri Nets*, pages 102–121, Gjern, Denmark, 1991.