FI MU

Faculty of Informatics
Masaryk University Brno

Under-Approximation Generation using Partial Order Reduction

by

Luboš Brim
Ivana Černá
Pavel Moravec
Jiří Šimša

FI MU Report Series FIMU-RS-2005-04

February 2005

Under-Approximation Generation using Partial Order Reduction

Luboš Brim[*] brim@fi.muni.cz
Ivana Černá[†] cerna@fi.muni.cz
Pavel Moravec[*] xmoravec@fi.muni.cz
Jiří Šimša[†] xsimsa@fi.muni.cz

March 1, 2005

Abstract

We propose a new on-the-fly approach which combines partial order reduction with the under-approximation technique for falsification and verification of LTL$_{-X}$ properties. It uses sensitivity relation and modified ample conditions to generate a reduced state space that is not fully stutter equivalent to the original one and it checks the desired property using representatives. Widening of under-approximations is fully automatic and does not rely on any supporting mechanisms like theorem-provers or SAT solvers.

1 Introduction

The state explosion problem is still a major bottleneck in applying model checking to large real life systems. Numerous approaches have been proposed to fight the problem. Partial order reduction [9, 10, 18, 21] and abstraction [3, 5, 11] are certainly the two most dominant techniques. Both reduce the size of the model checking problem by reducing the number of states to be dealt with, hence *approximating* the original system.

While partial order reduction controls the branching, in the abstraction technique a set of states is represented by one abstract state. In both cases the size of the state space

[*]The author has been supported by the Grant Agency of Czech Republic grant No. 201/03/0509
[†]The author has been supported by the Academy of Sciences of the Czech Republic grant No. 1ET408050503

is reduced, however there is an important difference in terms of modelled behaviours. Partial order reduction gives a model that has less behaviours than the original one (it is an *under-approximation*), while the model reduced by abstraction can have either more behaviours (it is an *over-approximation*) [3, 14] or less behaviours [15, 16] depending on the abstraction technique considered.

Approximations, however, are not always accurate enough to determine whether the original system satisfies its specification. There is the possibility of *false negatives* (if the over-approximation contains violating behaviours that are not part of the system) and *false positives* (the under-approximation may not include violating behaviours that are possible in the system). Thus the result from approximating may be inconclusive.

Over-approximations often only work well for invariant properties, since liveness properties may be erroneously invalidated by one of the extra behaviours. This highlights a drawback of an over-approximation, as it can add behaviours that invalidate a property in the abstract system while it is true in the concrete one. These spurious errors must then be removed by constraining the over-approximation (refining it). Under-approximations, on the other hand, are often used during model checking of liveness properties. This is because it can often be shown that the behaviours removed by under-approximation do not influence the verification result. Of course, validity of the property in the under-approximated model does not necessary mean the property is true in the original one and therefore another under-approximation has to be considered (typically a widening of the previous one).

In this paper, we concentrate on LTL_{-X} (LTL without the *next* operator) model checking. When reducing the model using partial order technique we end up with an exact approximation which is *stutter equivalent* to the model. This means that *all* LTL_{-X} properties are preserved in the reduced model. As the primary aim of under-approximations is falsification, we can achieve more massive reduction by lifting the stuttering equivalence requirement.

This work presents an approach that exploits partial order reduction techniques to compute a series of under-approximations in a fully automatic way. As already pointed out in [12] partial order reduction algorithms can be used to statically determine behaviours to be included into the subset of all behaviours.

Partial order reduction algorithm [4] computes the exact approximation by exploring only a subset of transitions, called an *ample set*, enabled at each state encountered during the state space generation. There are two factors that influence which transitions are selected: the given property (specification) through visibility of transitions and the verified model through independence of its transitions. Ample sets are determined using suitable conditions (C0 to C3). We propose two orthogonal ways to generate a strictly smaller subset of behaviours than generated by the partial order reduction. The first one uses generalisation of invisibility to determine the set of behaviours in the under-approximation and works with original conditions C0 to C3. The second one is based on a suitable modification of the condition C3 and can be combined with the first method as well. As the term invisibility strictly relates to propositions in the specification only, we use the notion of *insensitivity* to refer to transitions that are considered. In fact, insensitivity will guide the successive generation of under-approximations in our approach. In both techniques, if the property cannot be falsified on the received under-approximation we extend the set of sensitive transitions and compute the next under-approximation. The procedure works on-the-fly and is guaranteed to terminate as we eventually generate representatives of all possible behaviours.

There are three important points to note. First, the series of under-approximations is not monotonic with respect to set inclusion of behaviours (strictly speaking we are not computing a widening of the previous under-approximation). However, it is guaranteed that successive under-approximations *represent* monotonically growing set of behaviours of the original model, which is a distinguished feature of our approach. Second, we can choose in advance a monotonic sequence of sets of insensitive transitions, hence put an upper bound on the number of iterations to be performed in the worst case. Third, the generation of under-approximations is guided purely by a syntactical information from the system description and no other support, e.g. theorem prover or SAT solver, is required.

Our technique is related to the approach proposed in [12]. In both cases the under-approximation is given by partially expanding some states. The difference is both in the way the subset of all enabled transitions is determined and in the way the widening is achieved. We discuss the differences in more detail in the related work section.

2 Partial Order Reduction and Under-Approximations

Assume, we are given a model $\mathcal{M}$ and an LTL_{-X} formula φ. Let M be the full state space of the model $\mathcal{M}$. A state space $M_U \subseteq M$ such that $M_U \not\models \varphi \implies M \not\models \varphi$ is called an *under-approximation*. An under-approximation $M_E \subseteq M$ such that $M_E \models \varphi \iff M \models \varphi$

is called an *exact approximation*. In order to resolve the model checking problem for $\mathcal{M}$ and φ, we construct a (finite) sequence of under-approximations $M_1, \dots, M_n$ such, that M_n is an exact approximation. In the case $M_i \not\models \varphi$ for some $i \leq n$ we stop the generation with the answer $M \not\models \varphi$. If there is no such i, the answer is $M \models \varphi$ (as M_n is an exact approximation).

Our approach exploits partial order reduction technique to generate under-approximations. Before describing the procedure itself we summarise basic concepts that play role in this paper, for more details see [4, 10, 18].

We consider asynchronous multi-process systems defined as a composition of individual processes following the standard interleaving semantics. The systems are formally modelled as state transition systems. A state transition system is defined as a tuple $M = (S, T, s_0, L)$, where S is a set of states, $s_0 \in S$ is an initial state, T is a set of transitions $\alpha \subseteq S \times S$, and $L : S \rightarrow 2^{AP}$ is a labelling function that assigns to each state a subset of some set *AP* of atomic propositions.

A transition $\alpha \in T$ is *enabled* in a state s iff there is a state s' such that $(s, s') \in \alpha$. The set of all transitions enabled in a state s is denoted *enabled(s)*. We presuppose that transitions are deterministic, i.e., for every α and s there is at most one s', denoted as $\alpha(s)$, with $(s, s') \in \alpha$. In this case we say that s' is a *successor* of s.

The partial order method exploits the fact that the transitions can be executed concurrently and interleaved in either order. This can be formalised by defining an independence relation on pairs of transitions that can execute concurrently.

Definition 2.1 (indepedence). *An* independence *relation* $I \subseteq T \times T$ *is a symmetric and anti-reflexive relation, satisfying the following two conditions for each state* $s \in S$ *and for each* $(\alpha, \beta) \in I$*:*

1. *Enabledness – If* $\alpha, \beta \in \mathtt{enabled}(s)$ *then* $\alpha \in \mathtt{enabled}(\beta(s))$.
2. *Commutativity – If* $\alpha, \beta \in \mathtt{enabled}(s)$ *then* $\alpha(\beta(s)) = \beta(\alpha(s))$.

The dependency *relation is the complement of* I.

Heuristic methods are utilised for an efficient computation of the dependence relation according to the conditions mentioned above.

The independence relation suggests a potential reduction to the state transition system by selecting only one from the independent transitions originating from a state s. However, this cannot guarantee that the reduced state transition system is a correct replacement of the full one as it does not take into account the property to be checked.

Also, eliminating one of the states $\alpha(s)$ or $\beta(s)$ may cause some of its successors (which may be significant for the verification) not to be explored. This might be advantageous when building under-approximations but, on the other hand, we would like to build a "faithful" approximation reflecting the checked property. Therefore, additional conditions for the correctness of the reduction are needed.

First, the concept of *invisibility* of a transition formalises what it means that a property is taken into account.

Definition 2.2 (visibility). *A transition $\alpha \in T$ is* invisible *with respect to a set of propositions $AP' \subseteq AP$ if for each pair of states $s, s' \in S$ such that $\alpha(s, s')$, $L(s) \cap AP' = L(s') \cap AP'$ holds. A transition is* visible *if it is not invisible.*

The set AP' is usually induced by atomic propositions occurring in the verified formula. As the visibility relation is strictly related to some set of atomic proposition, we introduce a *sensitivity* relation to approximate the visibility relation.

Definition 2.3 (sensitivity). Sensitivity ρ *is a unary relation on the set of all transitions. A transition t is* sensitive, insensitive *if $t \in \rho$, $t \notin \rho$ respectively.*

Our approach relies on enriching the set of sensitive transitions, which is either subset or superset of the set of visible transitions. The reduced state transition system, denoted by M_R, is generated by a modified generation algorithm, which explores only a subset of transitions, called an *ample set*, enabled at each state encountered during the generation. The ample set can be defined in a manner that does not depend on the particular way the state transition system is generated. This is accomplished by a set of *conditions* relating the full state transition system to the corresponding reduced one. Note, that there could be more than one ample set satisfying the conditions for a given state. We say that a state s is *fully expanded* whenever $\mathsf{ample}(s) = \mathsf{enabled}(s)$.

Definition 2.4 (ample conditions). *Let AP' be a set of atomic propositions and ρ a sensitivity relation.* Ample conditions *with respect to ρ are:*

C0 $\mathsf{ample}(s) = \emptyset$ *iff* $\mathsf{enabled}(s) = \emptyset$.

C1 *Along every path in the full state graph M that starts at s, the following condition holds: a transition that is dependent on a transition in $\mathsf{ample}(s)$ cannot be executed without a transition in $\mathsf{ample}(s)$ occurring first.*

C2 *If $\mathsf{enabled}(s) \neq \mathsf{ample}(s)$, then every $\alpha \in \mathsf{ample}(s)$ is* ***insensitive****.*

C3 (cycle closing condition) *A cycle in the reduced state graph* M_R *is not allowed if it contains a state in which some transition* α *is enabled, but is never included in* $\mathtt{ample}(s)$ *for any state* s *on the cycle.*

Note that for a sensitivity relation which agrees with visibility (i.e., every visible transition is sensitive) these conditions characterise the ample sets needed to generate the reduced state transition systems sufficient for checking safety and liveness properties. The reduced state transition system is in this case an exact approximation of the original one with respect to LTL_{-X} properties.

While the conditions **C0**, **C1**, and **C2** can be checked locally, the condition **C3** is a global one and in practise the condition **C3** is checked *in constant time* using a *proviso*. An example of such a proviso is the condition **C3-one**, which can be used during a depth-first search based generation of the state space.

C3-one If a state s is not fully expanded, then no transition in $\mathtt{ample}(s)$ leads to a state on the search stack.

Our aim is to construct an under-approximation which is as small as possible. The first possibility is to approximate the visibility relation by the sensitivity relation. Another possibility is to weaken the condition **C3**. To this end we introduce a new condition $\widetilde{\mathbf{C3}}$.

$\widetilde{\mathbf{C3}}$ From all states in the reduced system M_R there is a finite path in M_R leading to a fully expanded state.

By replacing the condition **C3** with the condition $\widetilde{\mathbf{C3}}$ we are able to preserve certain behaviours. In the next section we give a precise characterisation of such behaviours, justifying thus formally our approach.

In order to check $\widetilde{\mathbf{C3}}$ efficiently a new proviso is needed. The proviso we use requires that a state is fully expanded whenever *all* successor states are already on the stack. The relation between $\widetilde{\mathbf{C3}}$ and the new proviso is proved in Section 4.

$\widetilde{\mathbf{C3}}$**-all** If a state s is not fully expanded, then not all transitions in $\mathtt{ample}(s)$ lead to a state on the search stack.

Not only does this proviso allow for under-approximation, but it also yields better reductions of state spaces. In particular, in models with many cycles almost every possible ample set contains an edge leading back to a search stack enforcing the full expansion under the proviso **C3-one**. The new proviso can deliver substantial reduction in such cases.

Now we are ready to sketch how the partial order method can be employed for computing under-approximations. Let us assume we are given a model $\mathcal{M}$ and a LTL$_{-X}$ formula φ and an algorithm, which generates a (reduced) state space M_R of the model $\mathcal{M}$ such that conditions **C0** through either **C3** or $\widetilde{\mathbf{C3}}$ hold. Moreover, let us assume the sensitivity relation is initially empty. The state space M_R is obviously an under-approximation. We model-check the formula φ on M_R (in fact we do this on-the-fly during the state space generation). If M_R is an exact approximation or $M_R \not\models \varphi$, we are done. Otherwise, we mark an insensitive transition t as a sensitive one and recompute the reduced state space. The procedure is guaranteed to terminate as it eventually generates an exact approximation. We present two algorithms implementing this framework as well as possible strategies for widening of under-approximations in Section 4.

3 Preservation of Properties under $\widetilde{\mathbf{C3}}$

In this section we show how the reduced state space relates to the original one under the conditions **C0** through $\widetilde{\mathbf{C3}}$. Throughout this section we assume that every visible transition is also a sensitive one.

Let $M = (S, T, s_0, L)$ be a state transition system and ρ a sensitivity relation. A *path* from a state s in M is a finite or infinite sequence $\sigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots$ such that $s = s_0$ and for every i, $(s_{i-1}, s_i) \in \alpha_i$. The *length* of a finite path σ, denoted $|\sigma|$, is the number of its transitions.

Let η be a finite and σ a finite or infinite path. Then *first*(σ) is the first state of σ and *last*(η) is the last state of η. If *last*(η) = *first*(σ) then $\eta \circ \sigma$ denotes the *concatenation* of the paths. For a path σ let *trace*(σ) denotes the sequence of transitions in σ and *sens*(σ) denotes the sequence of sensitive transitions in σ. The length of a sequence of transitions γ is denoted $|\gamma|$.

Definition 3.1 (stuttering equivalence). *Two infinite paths* $\sigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots$ *and* $\eta = r_0 \xrightarrow{\beta_1} r_1 \xrightarrow{\beta_2} \ldots$ *are* stutter equivalent, *denoted* $\sigma \sim_{st} \eta$, *if there are two infinite sequences of integers* $0 = i_0 < i_1 < i_2 < \ldots$ *and* $0 = j_0 < j_1 < j_2 < \ldots$ *such that for every* $k \geq 0$, $L(s_{i_k}) = L(s_{i_k+1}) = \ldots L(s_{i_{k+1}-1}) = L(r_{j_k}) = L(r_{j_k+1}) = \ldots L(r_{j_{k+1}-1})$.

Theorem 3.2. *Let* M *be a full state transition system and* M_R *be a reduced system satisfying the conditions* **C0** *through* $\widetilde{\mathbf{C3}}$. *Then for each path* σ *in* M *such that* $|sens(\sigma)| = \infty$ *there is a path* η *in* M_R *such that* $\sigma \sim_{st} \eta$.

There are two key steps to prove Theorem 3.2. The first one is the observation that for the equivalence only paths without so called *scattered cycles* are important. Informally, a scattered cycle on a path consists of those transitions which do not influence the validity of LTL_$_X$formulae over the path.

Definition 3.3 (scattered cycle). *Let* $M = (S, T, s_0, L)$ *be a state transition system and* $\gamma = \gamma_1 \cdot \gamma_2 \cdot \ldots \cdot \gamma_n$ *be a sequence of insensitive transitions from* T. *We say that* γ *is an* enabled cycle *if for all* $s \in S$ *such that the transition* γ_i *is enabled in the state* $\gamma_{i-1}(\ldots(\gamma_1(s))\ldots)$, $i = 1, \ldots, n$, *the equality* $\gamma_n(\ldots(\gamma_1(s))\ldots) = s$ *holds.*

We say that a path σ *in* M *contains a* scattered cycle γ *if and only if* γ *is an enabled cycle and there are paths* $\theta_1, \ldots, \theta_{n+1}$ *such that*

$\sigma = \theta_1 \circ (last(\theta_1) \xrightarrow{\gamma_1} first(\theta_2)) \circ \theta_2 \circ \ldots \circ \theta_n \circ (last(\theta_n) \xrightarrow{\gamma_n} first(\theta_{n+1})) \circ \theta_{n+1}$

and for all $i = 1, \ldots, n$

- *the transition* γ_i *is enabled in the state* $\gamma_{i-1}(\ldots(\gamma_1(first(\theta_1)))\ldots)$ *and*
- *all transitions in* $\theta_1, \theta_2, \ldots \theta_i$ *are independent on the transition* γ_i.

Lemma 3.4. *For each path* σ *in* M *with* $|sens(\sigma)| = \infty$ *there is an infinite path* σ' *in* M *such that* $\sigma \sim_{st} \sigma'$, $first(\sigma) = first(\sigma')$ *and* σ' *does not contain any scattered cycle.*

Proof: Let us suppose that σ contains a scattered cycle $\gamma = \gamma_1 \cdot \gamma_2 \cdot \ldots \gamma_n$ and $\sigma = \theta_1 \circ (last(\theta_1) \xrightarrow{\gamma_1} first(\theta_2)) \circ \theta_2 \circ (last(\theta_2) \xrightarrow{\gamma_2} first(\theta_3)) \circ \ldots \circ \theta_n \circ (last(\theta_n) \xrightarrow{\gamma_n} first(\theta_{n+1})) \circ \theta_{n+1}$.

According to the definition of the scattered cycle, the transition γ_2 is enabled in the state $first(\theta_2)$ and is independent on all transitions in θ_2. Therefore there is a path in M containing the scattered cycle γ and such that the transition γ_2 proceeds all transition from θ_2. Using the same argument iteratively we can conclude that there is a path $\theta_1 \circ (last(\theta_1) \xrightarrow{\gamma_1} \ldots \xrightarrow{\gamma_n} last(\theta_1)) \circ \theta'_2 \circ \ldots \theta'_n \circ \theta_{n+1}$ in M where $trace(\theta_i) = trace(\theta'_i)$ for all $i = 2, \ldots, n$. As γ is a scattered cycle, $\theta_1 \circ \theta'_2 \circ \ldots \theta'_n \circ \theta_{n+1}$ is a path in M stutter equivalent to σ. In this way we could iteratively remove all scattered cycles appearing in σ. However, by removing a scattered cycle from a path we could introduce to this path a new scattered cycle. To prove the existence of stutter equivalent path without scattered cycles we have to consider all existing and possible scattered cycles on the path σ simultaneously.

Let $\delta = \delta_1 \cdot \delta_2 \cdot \ldots$ be a (finite or infinite) subsequence of *trace*(σ) such that either δ_i is a transition of a scattered cycle in σ or there is a finite number of scattered cycles which can be removed from σ (through the above mentioned transformation) and δ_i becomes a transition of a scattered cycle in the resulting path.

Let $\alpha_1 \cdot \alpha_2 \cdot \ldots$ be a sequence of transitions which remains in *trace*(σ) after removing the subsequence δ. We need to prove that there is an infinite path σ' in M such that *first*$(\sigma) =$ *first*(σ') and *trace*$(\sigma') = \alpha_1 \cdot \alpha_2 \cdot \ldots$ These guarantee $\sigma \sim_{st} \sigma'$.

To prove that σ' is a path in M it is sufficient to prove that

$$\alpha_i \in \mathtt{enabled}(\alpha_{i-1}(\ldots(\alpha_1(\mathit{first}(\sigma)))\ldots))$$

for all i. Let δ_j occurs in σ before α_i. Then δ_j can be removed from the path (together with the scattered cycle it belongs to) and α_i still remains enabled thanks to the arguments mentioned above.

As *sens*$(\sigma) =$ *sens*(σ') and $|\mathit{sens}(\sigma)| = \infty$, the path σ' is infinite. □

The second step to prove Theorem 3.2 is the construction of stutter equivalent paths. Let σ be a path in M without any scattered cycle. We inductively describe a sequence of paths $\pi_0, \pi_1, \pi_2, \ldots$, where for every i, $\pi_i = \eta_i \circ \theta_i$ is a path in M, η_i is a path in M_R, and $|\eta_i| = i$. The path η, which is stutter equivalent to σ, is then defined as the limit of the sequence (η_i).

Basic step Let η_0 be an empty path and $\theta_0 = \sigma$.

Inductive step Let $s_0 = \mathtt{last}(\eta_i) = \mathtt{first}(\theta_i)$, $\theta_i = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \ldots$

There are two possibilities:

A If $\alpha_1 \in \mathtt{ample}(s_0)$ then $\eta_{i+1} = \eta_i \circ (s_0 \xrightarrow{\alpha_1} s_1)$, $\theta_{i+1} = s_1 \xrightarrow{\alpha_2} s_2 \ldots$

B The case $\alpha_1 \notin \mathtt{ample}(s_0)$ divides into two sub-cases.

B1 There is k such that $\alpha_k \in \mathtt{ample}(s_0)$ and (α_j, α_k) are independent for all $1 \leq j < k$. Then $\eta_{i+1} = \eta_i \circ (s_0 \xrightarrow{\alpha_k} \alpha_k(s_0))$. As transitions α_j are independent, $\alpha_k(s_0) \xrightarrow{\alpha_1} \alpha_k(s_1) \xrightarrow{\alpha_2} \alpha_k(s_2) \ldots$ is a path in M and according to Lemma 3.4 there is a path δ in M without scattered cycles and stutter equivalent to $\alpha_k(s_0) \xrightarrow{\alpha_1} \alpha_k(s_1) \xrightarrow{\alpha_2} \alpha_k(s_2) \ldots$ Let $\theta_{i+1} = \delta$.

B2 $\alpha_k \notin \mathtt{ample}(s_0)$ for any k. Then from the condition **C1** all transitions in $\mathtt{ample}(s_0)$ are independent on all transitions in θ_i. Let ξ be the shortest path in M_R from s_0 to a fully expanded state (the existence of such a path is guaranteed by $\widetilde{\mathbf{C3}}$) and let β be the first transition of ξ. Then $\eta_{i+1} = \eta_i \circ (s_0 \overset{\beta}{\to} \beta(s_0))$, $\theta_{i+1} = \beta(s_0) \overset{\alpha_1}{\to} \beta(s_1) \overset{\alpha_2}{\to} \beta(s_2) \ldots$

Cases **B1** and **B2** cover all possibilities which match up with **C1**.

To prove Theorem 3.2 we first characterise properties of the path η and then prove the stuttering equivalence.

Properties of η

Lemma 3.5. *For every* i, $\pi_i = \eta_i \circ \theta_i$ *is a path in* M, η_i *is a path in* M', *and* $|\eta_i| = i$.

Proof: By induction. Induction basis for $i = 0$ holds trivially. In induction step, we first prove that π_i is a path in M. It obviously holds for the case **A**. In the case **B1**, (α_j, α_k) are independent, for all $j < k$. Hence there is a path $\xi = s_0 \overset{\alpha_k}{\to} \alpha_k(s_0) \overset{\alpha_1}{\to} \alpha_k(s_1) \overset{\alpha_2}{\to} \ldots \overset{\alpha_{k-1}}{\to} \alpha_k(s_k) \overset{\alpha_{k+1}}{\to} s_{k+2} \overset{\alpha_{k+2}}{\to} \ldots$ in M, where α_k is moved before $\alpha_1\alpha_2\alpha_3 \ldots \alpha_{k-1}$. Note that $\alpha_k(s_k) = s_{k+1}$. Therefore, $\alpha_k(s_k) \overset{\alpha_{k+1}}{\to} s_{k+2}$ is the same as $s_{k+1} \overset{\alpha_{k+1}}{\to} s_{k+2}$. The correctness of removing scattered cycles is ensured by Lemma 3.4. In the case **B2** we execute a transition independent on all transitions in θ_{i-1}, hence θ_i is obviously a path in M.

Facts that η_i is a path in M' and $|\eta_i| = i$ are obvious in all cases, because we append to η_{i-1} exactly one transition from $\mathtt{ample}(\mathit{last}(\eta_{i-1}))$. □

Lemma 3.6. *Let* $\eta = \lim_{i\to\infty} \eta_i$. *Then* η *is a path in* M_R.

Proof: By induction to i. □

Lemma 3.7. *For every* i, θ_i *does not contain any scattered cycle.*

Proof: By induction to i. For $\theta_0 = \sigma$ the statement holds trivially. If θ_i is constructed applying **A** or **B2** it does not contain any scattered cycle as θ_{i-1} does not contain any. In case of **B1**, the desired property is enforced explicitly. □

Stuttering equivalence

Lemma 3.8. *The following holds for all* i, j *such that* $j \geq i \geq 0$.

1. $\pi_i \sim_{st} \pi_j$.

2. $\mathit{sens}(\pi_i) = \mathit{sens}(\pi_j)$.

3. *Let ξ_i be a prefix of π_i and ξ_j be a prefix of π_j such that sens(ξ_i) = sens(ξ_j). Then* $L(\mathtt{last}(\xi_i)) = L(\mathtt{last}(\xi_j))$.

Proof: It is sufficient to consider the case where $j = i + 1$. Consider three ways of constructing π_{i+1} from π_i. In case **A**, $\pi_{i+1} = \pi_i$ and the statement holds trivially.

In case **B1**, π_{i+1} is obtained from π_i by executing a insensitive transition α_k in π_{i+1} earlier than it is executed in π_i. In this case, we replace the sequence $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{k-1}} s_{k-1} \xrightarrow{\alpha_k} s_k$ by $s_0 \xrightarrow{\alpha_k} \alpha_k(s_0)) \xrightarrow{\alpha_1} \alpha_k(s_1) \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{k-1}} \alpha_k(s_{k-1}))$. Because α_k is insensitive, corresponding states have the same label, that is, for each $0 < l \leq k$, $L(s_l) = L(\alpha_k(s_l))$. Also, the order of the sensitive transitions remains unchanged. Possible deletion of scattered cycles has no impact to these properties. Parts 1, 2, and 3 follow immediately.

Finally, consider case **B2**, where the difference between π_i and π_{i+1} is that π_{i+1} includes an additional insensitive transition β. Thus, we replace some suffix $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \dots$ by $s_0 \xrightarrow{\beta} \beta(s_0)) \xrightarrow{\alpha_1} \beta(s_1) \xrightarrow{\alpha_2} \dots$. So, $L(s_l) = L(\beta(s_l))$ for $l \geq 0$. Again, the order of sensitive transitions remains unchanged and parts 1, 2, and 3 follow immediately. □

Lemma 3.9. *During the construction of η, the case* **A** *is chosen infinitely often.*

Proof: Let us assume that there is an index j such that for the construction of $\pi_j, \pi_{j+1}, \dots$ only the rule **B** is applied. Then either **B1** or **B2** is applied infinitely many times.

In case **B1** is applied infinitely many times there is an infinite sequence of transitions which are added to the prefix η_{j-1}. These transitions are insensitive and independent on all relevant transitions in θ_j. From finiteness of the set of states we have that some of the considered transitions form a enabled cycle, which is moreover a scattered cycle in θ_j. This contradicts Lemma 3.7.

This gives us an existence of an index $k \geq j$ such that for the construction of $\pi_k, \pi_{k+1}, \dots$ only the rule **B2** is applied. But this is a contradiction to the fact that in **B2** we always choose a transition from the shortest path to a fully expanded state. □

Lemma 3.10. *Let α be the first transition of θ_i. Then there exists $j > i$: α is the last transition of η_j and $\forall k : i \leq k < j$: α is the first transition of θ_k.*

Proof: The rules **B1** and **B2** leave the first transition α of θ_i unchanged, the rule **A** shifts the transition α to η_i. Thus it is sufficient to prove that during the construction of η, the rule **A** is applied infinitely often. This follows from Lemma 3.9. □

Lemma 3.11. *Let δ be the first sensitive transition on θ_i, prefix$_\delta(\theta_i)$ be the maximal prefix of trace(θ_i) that does not contain δ. Then*

either *δ is the first transition of θ_i and the last transition of η_{i+1}*

or
- *δ is the first sensitive transition of θ_{i+1} and*
  - *the last transition of η_{i+1} is insensitive and*
  - *$\mathtt{prefix}_\delta(\theta_{i+1})$ is a subsequence of $\mathtt{prefix}_\delta(\theta_i)$.*

Proof: If θ_{i+1} is constructed according to **A**, then δ is the last transition of η_{i+1}.

If **B1** is applied then an insensitive transition α_k from θ_i is appended to η_i to form η_{i+1} and δ is still the first sensitive transition of θ_{i+1}. The prefix $\mathtt{prefix}_\delta(\theta_i)$ is either unchanged or shortened by the transition α_k.

Otherwise an insensitive transition β is appended to η_i to form η_{i+1} and $\mathtt{prefix}_\delta(\theta_{i+1}) = \mathtt{prefix}_\delta(\theta_i)$. □

Lemma 3.12. *Let ν be a prefix of $sens(\sigma)$. Then there exists a path η_i such that $\nu = sens(\eta_i)$.*

Proof: By induction of the length of ν. The base holds trivially for $|\nu| = 0$. In the induction step we must prove that if $\nu\delta$ is a prefix of $sens(\sigma)$ and there is a path η_i such that $sens(\eta_i) = \nu$, then there is a path η_j with $j > i$ such that $sens(\eta_{i+1}) = \nu\delta$. Thus, we need to show that δ will be eventually added to η_j for some $j > i$, and that no other sensitive transition will be added to η_k for $i < k < j$. According to case **A** in the construction, we may add a sensitive transition to the end of η_k to form η_{k+1} only if it appears as the first transition of θ_k. Lemma 3.11 shows that δ remains the first sensitive transition in successive paths θ_k after θ_i unless it is being added to some η_j. Moreover, the sequence of transitions before δ can only shrink. Lemma 3.10 shows that the first transition in each θ_k is eventually removed and added to the end of some η_l for $l > k$. Thus, δ as well is eventually added to some sequence η_j. □

Proof: [of Theorem 3.2]

We will show that the described path $\eta = \lim_{i\to\infty} \eta_i$ is stutter equivalent to the original path σ.

First note that $sens(\sigma) = sens(\eta)$. It follows from Lemma 3.12 that for every prefix of σ there is a prefix of η with the same sequence of sensitive transitions. The opposite follows from Lemma 3.8.

Next we construct two infinite sequences of indexes $0 = i_0 < i_1 < \ldots$ and $0 = j_0 < j_1 < \ldots$ that define corresponding stuttering blocks of σ and η, as required in Definition 3.1. For every natural n, let i_n be the length of the smallest prefix ξ_{i_n} of σ that contains exactly n sensitive transitions. Let j_n be the length of the smallest prefix η_{j_n} of η

that contains the same sequence of sensitive transitions as ξ_{i_n}. Recall that η_{j_n} is a prefix of π_{j_n}. Then by Lemma 3.8, $L(s_{i_n}) = L(r_{j_n})$. By the definition of sensitive transitions we also know that if $n > 0$, for $i_{n-1} \leq k < i_n - 1$, $L(s_k) = L(s_{i_{n-1}})$. This is because i_{n-1} is the length of the smallest prefix $\xi_{i_{n-1}}$ of σ that contains exactly $n-1$ sensitive transitions. Thus, there is no sensitive transition between i_{n-1} and $i_n - 1$. Similarly, for $j_{n-1} \leq l < j_n - 1$, $L(r_l) = L(r_{j_{n-1}})$. □

4 Algorithms and Experiments

In this section we describe two model checking algorithms we have designed for LTL_{-X} model checking using under-approximations. Although both of them have similar structure, they differ in handling the widening of under-approximations, in the identification of the exact approximation, and in the conditions they use for the state space generation.

The first algorithm (see Figure 1) is based on the original set of conditions **C0** through **C3**. In each iteration it picks at random an insensitive visible transition and marks it as sensitive. Clearly, after a finite number of iterations we reach a situation, where all insensitive transitions are invisible and thus the reduced state space is stutter equivalent to the original one (i.e., it is an exact approximation).

```
funct $A(\mathcal{M}, \varphi)$
  compute the set $\mathcal{V}$ of visible transitions using $\mathcal{M}$ and $\varphi$;
  the set $\mathcal{S}$ of insensitive transitions is initially empty;
  compute the independence relation $\mathcal{D}$ using $\mathcal{M}$;
  while (true) do
        $M_R = Generate(\mathcal{M}, \mathcal{D}, \mathcal{S})$;
        if ($\varphi \not\models M_R$) then return false; fi
        if ($\mathcal{S} == \mathcal{V}$) then return true; fi
        let $\alpha$ be a random transition from $\mathcal{V} \setminus \mathcal{S}$;
        mark transition $\alpha$ as a sensitive;
  od
```

Figure 1: Under-approximation algorithm $A(\mathcal{M}, \varphi)$

The second algorithm (see Figure 2) is based on the conditions **C0** through $\widetilde{\mathbf{C3}}$. It maintains a set C of transitions, which lie on some *insensitive cycle* (i.e., all transition on the cycle are insensitive) in labelled transition system of some process (recall that

```
funct B($\mathcal{M}$, $\varphi$)
  compute the set $\mathcal{V}$ of invisible transitions using $\mathcal{M}$ and $\varphi$;
  the set $\mathcal{S}$ of insensitive transitions is initially empty;
  compute the independence relation $\mathcal{D}$ using $\mathcal{M}$;
  while ($\mathcal{S} \neq \mathcal{V}$) do
        $M_R = Generate\text{-}2(\mathcal{M}, \mathcal{D}, \mathcal{S})$;
        if ($\varphi \not\models M_R$) then return false; fi
        let $\alpha$ is a random transition from $\mathcal{V}$;
        mark transition $\alpha$ as a sensitive;
  od
  compute the set C;
  while (true) do
        $M_R = Generate\text{-}2(\mathcal{M}, \mathcal{D}, \mathcal{S})$;
        if ($\varphi \not\models M_R$) then return false; fi
        if ($C == \emptyset$) then return true; fi
        let $\alpha$ is a random transition from C;
        mark transition $\alpha$ as a sensitive;
        recompute the set C;
  od
```

Figure 2: Under-approximation algorithm B($\mathcal{M}$, φ)

we verify asynchronous multi-process systems). The generation of the next under-approximation is done in the following manner. It mimics the first algorithm until the condition $\mathcal{S} == \mathcal{V}$ holds. From that point on it iteratively picks at random a transition from the set C, marks it as a sensitive one, and updates the set C accordingly. As the number of all cycles in all processes is finite, after a finite number of iterations all cycles in all processes are sensitive. Consequently, every path in the full state space must contain an infinite number of sensitive transitions and due to Theorem 3.2 it has a stutter equivalent path in the reduced state space (i.e., the reduced state space is an exact approximation).

Whereas there are many known ways of how to resolve the model checking problem, it might not be obvious how to generate a reduced state space fulfilling the conditions **C0** through $\widetilde{\mathbf{C3}}$. Therefore we present a pseudo-code for the algorithm Generate-2 (see Figure 3). Note that in our implementation we perform LTL_X model checking on-the-fly during the state space generation.

The algorithm takes as input a model $\mathcal{M}$, an independence relation $\mathcal{I}$, and a set of sensitive transitions $\mathcal{S}$. It returns a reduced state space. The algorithm uses boolean

```
funct Generate-2($\mathcal{M}, \mathcal{I}, \mathcal{S}$)
  begin
    let $s_0$ be an initial state;
    $stack.push(s_0)$;
    while $(!stack.empty())$ do
          $state = stack.top();\ expand(state);\ stack.pop();$
    od
    return the traversed state space;
  where
  funct $expand(state)$
    $valid = false;$
    let $\mathcal{T}$ is a set of all proper subsets of $enabled(s)$ satisfying C0, C1 and C2;
    foreach $T \in \mathcal{T}$ do
      foreach $\alpha \in T$ do
        if $(!on\_stack(\alpha(state)))$ then $valid = true;$ fi
      od
      if $(valid)$ then $ample = T;$ break; fi
    od
    if $(!valid)$ then $ample = enabled(state);$ fi
    foreach $\alpha \in ample$ do
      if $(!visited(\alpha(state)))$ then $stack.push(\alpha(state));\ expand(\alpha(state));$ fi
    od
  end
```

Figure 3: Algorithm Generate-2($\mathcal{M}, \mathcal{I}, \mathcal{S}$)

functions $\mathit{on\_stack}$ and $\mathit{visited}$ to traverse the state space in the depth-first search manner. Given a state, the function $\mathit{on\_stack}$ returns true iff the state is on the stack and the function $\mathit{visited}$ returns true iff the state is on the stack or it was on the stack in the past.

Lemma 4.1. *The algorithm* $\mathsf{Generate\text{-}2}(\mathcal{M}, \mathcal{I}, \mathcal{S})$ *generates a reduced state space fulfilling the conditions* ***C0*** *through* $\widetilde{\mathbf{C3}}$*.*

Proof: Conditions **C0** , **C1** and **C2** can be checked locally and their verification is an implicit part of the algorithm (line 12). The condition $\widetilde{\mathbf{C3}}$ is ensured using the stack. We show by induction on the number of states removed from the stack that $\widetilde{\mathbf{C3}}$ holds for every state.

Basic step Let s be the first state to be removed from the stack. By simple argument it follows that all of its successors are on the stack. Therefore the state s is fully expanded as the condition on line 15 does not hold for any successor of s.

Inductive step Let s_i be the i-th state to be removed from the stack. We want to show that if a fully expanded state is reachable from all $s_1, \ldots, s_n$ then a fully expanded state is reachable also from s_{n+1}. Again, by simple argument it follows that all successors of the state s_{n+1} are either backtracked or on the stack. There are two cases:

a) All successors of the state s_{n+1} are on the stack. Similar arguments as in the basic step can be used to to show that s_{n+1} is fully expanded.

b) There is a successor of s_{n+1} which is not on the stack. Such a successor has been removed from the stack and by the induction hypothesis, there is a fully expanded state reachable from it. □

Experiments

We have implemented both under-approximation algorithms. The implementation has been done in C++ using tools provided by our own distributed verification environment DiVinE and the experiments have been performed on the Intel Pentium 4 2.6 GHz workstation with 1 GB of RAM.

For evaluation of the algorithm $A(\mathcal{M}, \varphi)$ (Figure 1), which approximates the visibility relation through the sensitivity relation, we have considered three different models

Model	**Sat.**	**Full**	**C3**	$A(\mathcal{M},\varphi)$	**Iter.**	**Ratio**	
Peterson-6err	no	5781294	5781294	3108806	1	100%	54%
LossyProtocol	no	1008383	938983	510286	1	93%	51%
Philo10L	yes	100014	100014	100014	20	100%	100%

Table 1: Experiments for the under-approximation of visibility

– erroneous version of the Peterson's algorithm, communication protocol with lossy channels, and dining philosophers. The results are summarised in Table 1.

The first three columns identify the type of the model, the validity of the property being checked, and the size of the full state space respectively. The fourth column gives the size of the state space reduced using the original partial order reduction [18]. The next two columns give the size of the state space and the number of iterations needed by our algorithm in order to resolve the model checking problem. Finally, the last column gives the reduction ratio of both reduction techniques with respect to the size of the full state space. For example, to falsify the property the original partial order method generates 93% of the full state space while our algorithm generates 51% only.

For evaluation of the algorithm $B(\mathcal{M},\varphi)$ (Figure 2), which replaces the condition **C3** with $\widetilde{\mathbf{C3}}$, we have considered two models for mutual exclusion – the Peterson's and token algorithm. As we wanted to focus on the influence of $\widetilde{\mathbf{C3}}$ only, we have started the first iteration with the sensitivity relation equal to the visibility relation. The results are summarised in Table 2.

Model	**Sat.**	**Full**	**C3**	$B(\mathcal{M},\varphi)$	**Iter.**	**Ratio**	
Peterson-4	no	262601	216069	207197	1	82%	79%
Peterson-4	yes	262598	211872	262598	4	81%	100%
Peterson-4L	no	132117	132117	103963	1	100%	79%
Token-13	yes	167936	52	167936	13	$3 \cdot 10^{-2}$%	100%
Token-13L	no	167936	167936	52	1	100%	$3 \cdot 10^{-2}$%

Table 2: Experiments for the condition $\widetilde{\mathbf{C3}}$

The first four columns give the same information as in Table 1. The next two columns give the size of the state space and the number of iterations needed by the algorithm in order to resolve the model checking problem. Finally, the last column has the same meaning as in Table 1.

As demonstrated by experiments, both under-approximation methods work very well in detecting property violations as the computation terminates after the first few iterations (in our experiments always after the first one). Consequently, the size of the generated state space was smaller comparing to the one generated by the original partial order reduction.

The first method, based on the condition **C3**, is aimed at models with many visible transitions. The second method, based on the condition $\widetilde{\mathbf{C3}}$, performs well for models where the partial order method with the condition **C3** does not reduce the state space significantly.

In the case the verified property is satisfied, both methods have disadvantages typical for all approximation methods. The time needed for computing an exact approximation might be even higher than the time needed for computing the full state space.

Note that as both presented methods are based on partial order reduction, their success depends heavily on the dependence relation of the model. One can try to approximate the dependence relation in a similar way we have approximated the visibility relation. However, this would push the method out of the partial order reduction framework. Therefore we have not investigated this possibility.

5 Related Work and Conclusions

In this paper we have proposed a new on-the-fly approach which combines partial order reduction with the under-approximation technique for validation of LTL_{-X} properties. It uses sensitivity relation and modified cycle closing condition to generate a reduced state space that is not fully stutter equivalent to the original one and it checks the desired property using representatives. To the best of our knowledge, the presented fully automatic approach to widening of under-approximations is the first one that does not rely on any supporting mechanisms like theorem-provers or SAT solvers.

Recently, several approaches for the verification that are based on under-approximations have been proposed. In [1] the authors integrate symmetry reduction and under-approximation with symbolic model checking. The main objective of the algorithm is falsification and provides verification only under certain conditions. The extended algorithm allows checking both safety and liveness properties, however, is not on-the-fly.

Another approach, which is more closer to the technique suggested in this paper, is [12]. The presented procedure checks models with an increasing set of allowed inter-leavings of the given set of processes, starting from a single interleaving. The procedure relies on SAT solvers' ability to produce proofs of unsatisfiability, from these proofs it derives information that guides the process of adding inter-leavings on the one hand, and determines termination on the other. The under-approximation widening is fully automatic in contrast to previous solutions.

Yet another approach is currently being investigated [17]. It builds under-approximations using predicate abstractions and is dependent on a theorem-prover.

Our procedure has some similarities with the approach taken in [12]. Both methods are fully automatic and both are based on suitable partial expansions of enabled transitions. However, there are fundamental differences. While in [12] the under-approximations are monotonically widened, in our approach the individual under-approximations are generally different. Unlike [12], the subset of enabled transitions is not limited to one process, but several processes can perform their transitions. This gives the possibility for a more finer refinement. The other difference is that we do not need any external support to perform the procedure, the method can be easily incorporated into existing LTL model checking tools.

For future research, we would like to perform a more extensive study of practical behaviour of the algorithms. The condition $\widetilde{\mathbf{C3}}$ is not based on cycle detection, as it uses reachability only. This gives a good chance to be used in a distributed environment and we are currently investigating how to combine the new proviso with distributed partial order reduction and how to parallelise our procedure for under-approximations. The result achieved so far are quite promising.

References

[1] S. Barner and O. Grumberg. Combining Symmetry Reduction and Upper-Approximation for Symbolic Model Checking. In *Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 93–106. Springer, 2002.

[2] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. *J. ACM*, 50(5):752–794, 2003.

[3] E. M. Clarke, O. Grumberg, and D. E. Long. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, Massachusetts, 1999.

[5] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *ACM Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977.

[6] D. Dill and H. Wong-Toi. Verification of Real-Time Systems by Successive over and under Approximation. In *Computer Aided Verification (CAV'95)*, volume 939, pages 409–422. Springer, 1995.

[7] D. Dill and H. Wong-Toi. Approximations for Verifying Timing Properties. In *Theories and Experiences for Real-Time System Development*, volume 2 of *AMAST Series in Computing*, pages 147–176. World Scientific Publishing, 1994.

[8] C. Flanagan and P. Godefroid. Dynamic Partial-Order Reduction for Model Checking Software. In *ACM Symposium on Principles of Programming Languages (POPL'05)*, pages 110–121. ACM Press, 2005.

[9] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *LNCS*. Springer, 1996.

[10] P. Godefroid and P. Wolper. A Partial Approach to Model Checking. In *IEEE Symposium on Logic in Computer Science*, volume 110, pages 305–326. Academic Press, Inc., 1994.

[11] S. Graf and H. Saïdi. Construction of Abstract State Graphs with PVS. In *Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[12] O. Grumberg, F. Lerda, O. Strichman, and M. Theobald. Proof-Guided Underapproximation-Widening for Multi-Process Systems. In *ACM Symposium on Principles of Programming Languages (POPL'05)*, pages 122–131. ACM Press, 2005.

[13] G. J. Holzmann and D. Peled. An Improvement in Formal Verification. In *FORTE'94*, volume 6, pages 197–211. Chapman & Hall, Ltd., 1994.

[14] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1994.

[15] W. Lee, A. Pardo, J-Y. Jang, G. Hachtel, and F. Somenzi. Tearing Based Automatic Abstraction for CTL Model Checking. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'96)*, pages 76–81. IEEE Computer Society, 1996.

[16] A. Pardo and G. D. Hachtel. Incremental CTL Model Checking using BDD Subsetting. In *Design Automation Conference (DAC'98)*, pages 457–462. ACM Press, 1998.

[17] C. Pasareanu, W. Visser, and R. Pelánek. Concrete Model Checking with Abstract Matching and Refinement. Submitted, 2005.

[18] D. Peled. All from One, One from All: on Model Checking using Representatives. In *Computer Aided Verification (CAV'93)*, number 697 in LNCS, pages 409–423. Springer, 1993.

[19] D. Peled. Combining Partial Order Reductions with On-the-Fly Model-Checking. *Formal Methods in System Design*, 8(1):39–64, 1996.

[20] D. Peled. Ten Years of Partial Order Reduction. In *Computer Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 17–28. Springer, 1998.

[21] A. Valmari. Stubborn Sets for Reduced State Space Generation. In *Advances in Petri Nets 1990*, volume 483 of *LNCS*, pages 491–515. Springer, 1991.