Partial order reduction for state/event LTL with application to component-interaction automata

N. Beneš *, L. Brim, B. Buhnova, I. Černá, J. Sochor, P. Vařeková

Faculty of Informatics, Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic

ARTICLE INFO

Article history:
Received 20 May 2009
Received in revised form 14 January 2010
Accepted 15 February 2010
Available online 9 March 2010

Keywords:
State/event LTL
Partial order reduction
Formal verification
Model checking
Component-based systems
Component-interaction automata

ABSTRACT

Software systems assembled from a large number of autonomous components become an interesting target for formal verification due to the issue of correct interplay in component interaction. State/event LTL (Chaki et al. (2004, 2005) [1,2]) incorporates both states and events to express important properties of component-based software systems.

The main contribution of this paper is a partial order reduction technique for verification of state/event LTL properties. The core of the partial order reduction is a novel notion of stuttering equivalence which we call state/event stuttering equivalence. The positive attribute of the equivalence is that it can be resolved with existing methods for partial order reduction. State/event LTL properties are, in general, not preserved under state/event stuttering equivalence. To this end we define a new logic, called weak state/event LTL, which is invariant under the new equivalence.

To bring some evidence of the method's efficiency, we present some of the results obtained by employing the partial order reduction technique within our tool for verification of component-based systems modelled using the formalism of component-interaction automata (Brim et al. (2005) [3]).

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Increasing complexity in software development stimulates application of new techniques that help to deliver systems in shorter time and with lower costs. One such technique is the component-based development, that builds software systems out of prefabricated autonomous components, often developed with no knowledge of their deployment context. Under such conditions, interaction among components in the system becomes a crucial issue in the system correctness.

Verification of component-based systems. Similarly to communicating processes, interaction of components can be formalized in terms of labelled transition systems, representing communicational behaviour of the components, and correctness of the systems in a temporal logic. In practice, real systems are composed of a large number of components which are often independent of each other and run concurrently. In such cases, automated verification becomes challenging due to their size and complexity. This motivates the search of component-specific attributes, which can be exploited in order to make the verification feasible.

Correctness attributes. One of the crucial observations in verification of component interaction in component-based systems is that the correctness attributes often highlight interaction among specific components which form only a small part of the
system. Even if the rest of the system is also important as it may coordinate these components, with appropriate reduction techniques a large portion of its complexity could be abstracted away during verification.

Partial-order reduction technique. One of the techniques successfully employed to state-space reduction is the partial order reduction. This technique is able to identify redundancies in the model during the verification process, commonly caused by interleaving of independent actions. This allows the technique to omit the generation of some of them while at least one representative of each equivalence class remains part of the actually verified model.

State/event temporal logic. In component-based systems, as in any modular programs in general, communication among components proceeds via events, which represent message passing, service calls, delivery of return values, etc. At the same time, components preserve also persistent state information about current values of their attributes. The adequate logic formalizing properties of these systems hence should be able to express both state-based and action-based properties, as well as their combinations. Research conducted on this topic resulted in the state/event LTL [1,2]. For the logic, however, there is no partial order reduction method known at the time. The situation is complicated for its fragment, the action-based LTL, as well. Please note that in this paper, we use the terms action and event interchangeably.

Contribution. The main contribution of this paper is a partial order reduction method for state/event LTL, and for action-based LTL as its special case. The whole framework is moreover defined in a way that it can be turned at no additional cost into the standard partial order reduction problem for state-based LTL at the end. Hence, it can be resolved with known and widely implemented techniques.

For the reduction to have the required effect one needs to identify an equivalence of two runs. The equivalence should allow for considerable level of reduction, while preserving temporal properties that reflect meaningful correctness attributes for the studied systems. We define an equivalence relation, state/event stuttering equivalence, driven by the correctness attributes of component-based systems, highlighting only the interesting interaction of components as discussed above, and characterize the state/event LTL properties preserved by the equivalence in terms of a new logic named weak state/event LTL.

We have implemented weak state/event LTL with partial order reduction based on the method described in this paper. To bring some evidence about the efficiency of the method, we present some experimental results.

Outline of the paper. After a brief review of related work in Section 2, we define basic terms and structures in Section 3. The equivalence and the corresponding logic are introduced in Section 5, which is preceded by thorough motivation underlying their definition in Section 4. The partial order reduction technique is presented in Section 6, which is then followed by Section 7 presenting an implementation of the technique based on the formalism of component-interaction automata and the experimental results of an application in a case study. This is followed by a conclusion in Section 8.

2. Related work

A combination of state-based and action-based linear temporal logic, named state/event LTL, has been studied in [1,2]. The authors argue that formalisms including both states and actions are suitable for modelling modular systems, including component-based systems, better than pure state-based or action-based approaches. It is also shown that the automata-based verification method for state-based LTL [4] can be modified to a verification method for state/event LTL in a straightforward way and at no additional cost of time and space. As noted by the authors, the results indicate the importance of further research in reduction techniques. The partial order reduction is suggested as a future direction. Its need was also discovered in our recent work on verification of component-based systems [5,6].

The partial order reduction method was originally introduced in three independent works [7–9]. The approach has been further developed, but in connection with linear temporal properties, state-based LTL has always been assumed. The reason for leaving action-based and state/event LTL behind is most likely because the correctness of the partial order reduction method is based on the concept of stuttering invariance of properties [10]. To the best of our knowledge, the stuttering concept has currently no convenient analogue for neither state/event nor action-based LTL. There have been, though, approaches like [11] which solve this problem by transforming state/event systems into purely state-based ones and then using state-based LTL and the standard partial order reduction. However, this comes at a cost, in both enlarging the state space (the number of both states and transitions can be in the worst case multiplied by the size of the alphabet) and weakening the logic (LTL without the next operator is unable to distinguish between one and more consecutive executions of a single event in such a setting).

An approach that relates actions and stuttering equivalence is the temporal logic of actions [12], where the formulae are constructed in a way that they are stuttering invariant. However, the actions are formulated in terms of changes of state propositions and/or variables, not allowing an arbitrary concept of actions. In our approach, we adopt a more general attitude to actions, as we consider arbitrary actions not tied to the properties of states.

The idea behind the state/event stuttering equivalence defined in Section 5 bears many similarities to the concept of projection in [13]. However, the logic and the methods studied in [13] are more specialized and quite different from the general case of state/event LTL and partial order reduction.
3. Basic definitions

As a general modelling formalism for state/event systems we use labelled Kripke structures. Many automata-based approaches, such as Component-Interaction Automata [5,6], Interface Automata [14] and I/O automata [15], can be easily translated to labelled Kripke structures.

Definition 3.1 (LKS). A labelled Kripke structure (LKS) is a 6-tuple \( (S, \text{Act}, \Delta, s_{\text{init}}, A_p, \mathcal{L}) \) where \( S \) is a nonempty set of states, \( \text{Act} \) is a finite set of actions, \( \Delta \subseteq S \times \text{Act} \times S \) is a transition relation, \( s_{\text{init}} \in S \) is an initial state, \( A_p \) is a finite set of atomic propositions and \( \mathcal{L} : S \rightarrow 2^{A_p} \) is a state-labelling function. Instead of \( (s, a, s') \in \Delta \), we also write \( s \xrightarrow{a} s' \).

A run \( \pi \) of an LKS is an infinite alternating sequence of states and actions \( \pi = s_0, a_0, s_1, a_1, \ldots \) such that \( \forall i : s_i \xrightarrow{a_i} s_{i+1} \). We call a run initial if \( s_0 = s_{\text{init}} \). Given a run \( \pi \), we also define: the \( i \)th subrun of \( \pi \) as \( \pi^i = s_i, a_i, s_{i+1}, a_{i+1}, \ldots \), the \( i \)th state of \( \pi \) as \( \pi(i) = s_i \) and the \( i \)th action of \( \pi \) as \( \ell(\pi, i) = a_i \).

Other kinds of transition systems can be naturally translated into LKSs. The most commonly used are the Kripke structures, which correspond to LKSs without transition labels (i.e. actions), and the labelled transition systems, which correspond to LKSs without atomic propositions and state labelling.

Example 3.2. Consider a component-based system implementing a simple photo gallery, modelled as an LKS in Fig. 2. The system consists of three components: Controller, Updater and Logger, with behavioural descriptions in Fig. 1. The interface of the system is formed by Controller, which inserts photos with captions into the gallery. When provided with a picture and its caption, Controller asks Updater to update information in the gallery via action \text{Updater}, which synchronizes with a corresponding action in \text{Updater}. In the model of the system in Fig. 2, \text{Updater} gains focus and starts the update. It saves the changes first and then asks Logger to log the information. Logger responds to the Log call via \text{Log} and then completes the operation. Concurrently with the \text{MakeLog} operation of Logger, \text{Updater} returns response to Controller, which after all takes all the components (and hence the system) to the initial setting.

To make the LKS model of this system in Fig. 2 complete, we need to add the set \( A_p \) of atomic propositions and the state labelling function \( \mathcal{L} \). In this case we choose the atomic propositions reflecting action enabledness. That is, \( A_p = \{ \mathcal{E}(a) \mid a \in \text{Act} \} \) where each \( \mathcal{E}(a) \) represents a state proposition with the meaning “action \( a \) is enabled in this state”. The labelling function is then given as \( \mathcal{L}(s) = \{ \mathcal{E}(a) \mid \exists s' : s \xrightarrow{a} s' \} \) associating with each state the actions enabled in that state. For example \( \mathcal{L}(\mathcal{A}) = \{ \mathcal{E}(\text{Picture}), \mathcal{E}(\text{Caption}) \} \).

We continue with defining the linear temporal logic that encompasses both state propositions and actions. This definition is equivalent to the definition in [1], it only slightly differs in notation.

Definition 3.3 (SE-LTL). Let \( \text{Act} \) be a set of actions, \( A_p \) a set of atomic propositions. The syntax of the \text{state/event LTL} (SE-LTL for short) formulae is defined inductively as

\[
\varphi ::= \mathcal{P}(a) \mid p \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \mathcal{U} \varphi_2 \mid \mathcal{X} \varphi
\]

where \( a \) ranges over \( \text{Act} \) and \( p \) ranges over \( A_p \).
Let $\pi$ be a run of an LKS. The semantics of SE-LTL for runs is defined as:

- $\pi \models \mathcal{P}(a)$ if and only if $\ell(\pi, 0) = a$
- $\pi \models p$ if and only if $p \in \mathcal{L}(\pi(0))$
- $\pi \models \neg \varphi$ if and only if $\pi \not\models \varphi$
- $\pi \models \varphi \land \psi$ if and only if $\pi \models \varphi$ and $\pi \models \psi$
- $\pi \models \varphi \mathcal{U} \psi$ if and only if there exists $k \geq 0$ such that $\pi^k \models \psi$ and for all $j < k$, $\pi^j \models \varphi$
- $\pi \models \mathcal{X} \varphi$ if and only if $\pi^1 \models \varphi$

Further, we say that an LKS $M$ satisfies $\varphi$, written as $M \models \varphi$, if for all initial runs $\pi$ of $M$, $\pi \models \varphi$.

**Example 3.4.** Consider the LKS from Example 3.2 in Fig. 2. Let $\mathcal{F} \varphi$ stand for $\mathcal{U} \varphi$ and $\mathcal{G} \varphi$ stand for $\neg \mathcal{F} \neg \varphi$. The property reflecting that an arbitrary number of pictures can be inserted into the album, can be stated in SE-LTL as $\mathcal{G} \mathcal{F} \mathcal{P}$(Picture).

An example of a property using state atomic propositions could be “whenever the action Picture becomes enabled, it is eventually executed”, which is expressible as $\mathcal{G} (\mathcal{E}$(Picture) $\Rightarrow$ $\mathcal{F} \mathcal{P}$(Picture)).

It has been demonstrated in [1] that the automata-based approach for state-based LTL verification (see e.g. [4]) can be straightforwardly transformed into an automata-based verification method for SE-LTL with no extra cost.

### 4. Motivation

As mentioned in the introduction, to cope with the enormous size of real system models consisting of a large number of components it is necessary to employ reduction methods. The aim of such methods is to generate a reduced state space instead of the complete one while ensuring preservation of all required properties. One of such methods, the partial order reduction method, exploits the redundancies in the system caused by concurrent interleaving. When dealing with state-based LTL properties, the partial order reduction technique is built on the concept of stuttering equivalence [10]. Two runs of a system are considered to be stuttering equivalent if the only difference between them lies in sequential repetitions of states with identical labelling. The partial order reduction method then ensures that for each run of the system there is a stuttering equivalent run in the reduced state space. The subset of LTL properties that are preserved by this equivalence can be characterized syntactically: they are exactly those properties that can be written without the $\mathcal{X}$ operator.

To apply the partial order reduction method to SE-LTL, we need first to find a suitable concept that would play the role of stuttering equivalence for the state/event case. The above mentioned stuttering equivalence cannot be employed, as it considers state labelling only.
The first idea is to transfer the stuttering concept to actions. In the stuttering equivalence, consecutive states labelled with the same atomic propositions are ignored. Let us therefore consider an equivalence that ignores consecutive transitions labelled with the same action and let us call this equivalence action stuttering. It implies that, for instance, two runs of the form $s_0, a, s_1, a, s_2, b, s_3, c, s_4, c, \ldots$ and $q_0, a, q_1, b, q_2, b, q_3, b, q_4, c, \ldots$ are action-stuttering equivalent. The advantage of this straightforward approach is that all formulae of action-based LTL not using the $X$ operator are preserved by this equivalence. However, there is a number of arguments against this choice.

It is obvious that the partial order reduction method does not preserve this action-stuttering equivalence. Fig. 3 shows a typical situation. If the transitions labelled with $a$ and $b$ are independent and one of them is invisible, the partial order reduction method traverses just one of the two runs $ab\ldots$ and $ba\ldots$. However, those two runs are not action-stuttering equivalent.

The problem is more fundamental. Consider a component-based system consisting of two components whose interaction is to be verified. Suppose we extend the system with an additional component that does not influence the communication of the original ones. A suitable substitution for stuttering equivalence should consider every run corresponding to the interaction behaviour of the original components equivalent regardless of interleaving with the third component. It is clear that the proposed action-stuttering equivalence does not satisfy this reasonable property.

We define a new equivalence, which, while still retaining the stuttering concept with respect to the state propositions, employs a different approach towards the transition labels (actions). This new equivalence enjoys the property that it is preserved by the partial order reduction method, thus allowing all the advantages of it. This comes at a cost. Contrary to state-based LTL, we do not have any syntactic characterization of SE-LTL formulae that are preserved by the new equivalence. However, we show that they can be elegantly described in terms of an adjusted weak version of SE-LTL.

5. State/event stuttering equivalence

The main idea of the equivalence is that some of the actions are regarded as interesting. Transitions with noninteresting actions are then overlooked by the equivalence. As we want to consider both actions and states, this idea is combined with the stuttering principle for state propositions, i.e. transitions which change state propositions we are interested in cannot be overlooked. In order to define the equivalence formally, we introduce the notions of a projection and a signature.

**Definition 5.1 (Projection, Signature).** Let $\pi = s_0, a_0, s_1, a_1, \ldots$ be a run of LKS $(S, \text{Act}, \Delta, s_{\text{init}}, \text{Ap}, \mathcal{L})$, let $\text{Act}' \subseteq \text{Act}$ and $\text{Ap}' \subseteq \text{Ap}$. Let $\tau$ be a new symbol, $\tau \notin \text{Act}$. A projection of $\pi$ onto $\text{Act}'$ and $\text{Ap}'$ is defined as

$$\text{pr}^{\text{Ap}'}_{\text{Act}'}(\pi) = E_0, b_0, E_1, b_1, \ldots$$

where $E_i = \mathcal{L}(s_i) \cap \text{Ap}', b_i$ is equal to $\tau$ whenever $a_i \notin \text{Act}'$ and $b_i = a_i$ otherwise.

Furthermore, a signature of $\pi$ with respect to $\text{Act}'$ and $\text{Ap}'$, denoted as $\text{sig}^{\text{Ap}'}_{\text{Act}'}(\pi)$ is defined as the (finite or infinite) alternating sequence of sets of atomic propositions and actions that arises from the projection of $\pi$ onto $\text{Act}'$ and $\text{Ap}'$ by replacing every maximal subsequence of the form $E_i, \tau, E_{i+1}, \tau, \ldots,$ where $E_i = E_{i+1} = \ldots$, with just $E_i$.

**Example 5.2.** Let $\pi = A, \text{Caption}, B, \text{Picture}, D, \text{Update}, E, \text{MakeSave}, F, \ldots$ and let $\text{Act}' = \{\text{Update}\}$ and $\text{Ap}' = \{\delta(\text{Picture})\}$. Then the projection of $\pi$ onto $\text{Act}'$ and $\text{Ap}'$ is $\text{pr}^{\text{Ap}'}_{\text{Act}'}(\pi) = \{\delta(\text{Picture})\}, \tau, \{\delta(\text{Picture})\}, \tau, \emptyset, \text{Update}, \emptyset, \tau, \ldots$ and its signature is $\text{sig}^{\text{Ap}'}_{\text{Act}'}(\pi) = \{\delta(\text{Picture})\}, \tau, \emptyset, \text{Update}, \emptyset, \ldots$.

**Definition 5.3 (State/Event Stuttering Equivalence).** Let $\pi$ and $\sigma$ be two runs, let $\text{Act}'$ be a set of actions, $\text{Ap}'$ a set of atomic propositions. We say that $\pi$ and $\sigma$ are state/event stuttering equivalent with respect to $\text{Act}'$ and $\text{Ap}'$, denoted as $\pi \equiv_{\text{Act}', \text{Ap}'}^\text{SE} \sigma$, if they have the same signatures, i.e. $\text{sig}^{\text{Ap}'}_{\text{Act}'}(\pi) = \text{sig}^{\text{Ap}'}_{\text{Act}'}(\sigma)$.

Two LKSs are said to be state/event stuttering equivalent with respect to $\text{Act}'$ and $\text{Ap}'$, if for each run of one LKS there is a state/event stuttering equivalent run of the other and vice versa.

Stuttering equivalence is a special case of state/event stuttering equivalence for $\text{Act}' = \emptyset$.

**Definition 5.4 (Weak SE-LTL).** Let $\text{Act}$ be a set of actions, $\text{Ap}$ a set of atomic propositions and let $\text{Act}' \subseteq \text{Act}$. We define the weak state/event LTL with respect to $\text{Act}'$ and SE-LTL for short, as follows. The syntax of the formulae is defined inductively as:

$$\varphi ::= \overline{\varphi} | p | \neg \varphi | \varphi_1 \land \varphi_2 | \varphi_1 \mathbf{U} \varphi_2 | \mathbf{X} \varphi | \varphi_1 \mathbf{U}_a \varphi_2$$

where $a$ ranges over $\text{Act}'$ and $p$ ranges over $\text{Ap}$.
Let $\pi$ be a run of an LKS. The semantics for runs is defined inductively as

\[
\begin{align*}
\pi \models \tilde{P}(a) &\iff \exists k \geq 0 \colon \ell(\pi, k) = a \text{ and } \forall j < k : \ell(\pi, j) \notin \text{Act}' \\
\pi \models p &\iff p \in \mathcal{L}(\pi(0)) \\
\pi \models \neg \varphi &\iff \pi \not\models \varphi \\
\pi \models \varphi \land \psi &\iff \pi \models \varphi \text{ and } \pi \models \psi \\
\pi \models \varphi \Rightarrow \psi &\iff \exists k \geq 0 : \pi^k \models \psi \text{ and } \forall j < k : \pi^j \not\models \varphi \\
\pi \models \tilde{X} \varphi &\iff \exists k \geq 0 : \ell(\pi, k) \in \text{Act}' \text{, } \forall j < k : \ell(\pi, j) \notin \text{Act}' \text{ and } \pi^{k+1} \models \varphi \\
\pi \models \varphi \Upsilon \psi &\iff \exists k \geq 0 : \ell(\pi, k) = a, \pi^{k+1} \models \psi \text{ and } \forall j < k + 1 : \pi^j \not\models \varphi
\end{align*}
\]

The main difference between SE-LTL and wSE-LTL is in the semantics of the next and action operators. While $\mathcal{P}(a)$ states that “the first action is $a$”, $\tilde{P}(a)$ states that “the first interesting action is $a$”. The formula $\tilde{X} \varphi$ states that “in the next step, $\varphi$ holds”, while the formula $\tilde{X} \varphi$ states that “after the next interesting action, $\varphi$ holds”. Actually, the definition of $\tilde{X} \varphi$ is of the form “there is a next interesting action and after this action, $\varphi$ holds”. We could also sometimes be interested in stating the property that “if there is a next interesting action, then after this action, $\varphi$ holds”. Nevertheless, this alternative $\tilde{X} \varphi$ operator can be defined as a derived operator: $\tilde{X} \varphi := \neg \tilde{X} \neg \varphi$.

Additionally, wSE-LTL has a new operator $\Upsilon$. The motivation is to express properties like “atomic proposition $p$ holds until action $a$ happens”. In SE-LTL, this can be expressed with $p \Upsilon (p \land \mathcal{P}(a))$. In wSE-LTL, this is no longer possible as the semantics of $\mathcal{P}(a)$ is different and the intuitive solution $p \Upsilon (p \land \mathcal{P}(a))$ holds even for runs that do not satisfy the original property, e.g. a run with signature $\{p\}$, $\tau$, $\{\neg p\}$, $a$, $\ldots$. Thanks to the $\Upsilon$ operator, this property is expressible as $p \Upsilon \text{true}$. The $\Upsilon$ operator is not needed in the action-based fragment of wSE-LTL. The reader may verify that every formula $\varphi \Upsilon \psi$, where $\varphi$ and $\psi$ both do not use state atomic propositions, is equivalent to $\varphi (\mathcal{P} (a) \land \varphi \land \tilde{X} \varphi)$. In the previous, interesting” means “from $\text{Act}$”. In many natural cases, the interpretation of a wSE-LTL formula remains the same regardless of the choice of $\text{Act}$ as long as $\text{Act}$ contains all action labels from $\mathcal{P}(a)$ and $\Upsilon$ subformulas of the formula. This is, however, not true in general, as semantics of some wSE-LTL formulae may depend on this choice. When specifying properties in wSE-LTL, it is therefore assumed that a pair $(\varphi, \text{Act})$ is given instead of just $\varphi$.

An example of a formula that has different validity depending on the choice of $\text{Act}$ is given in the following.

Example 5.5. Consider the LKS from Example 3.2 in Fig. 2 and the formula

\[
\mathcal{G}(\tilde{P}(\text{Update}) \Rightarrow \tilde{X}((\varepsilon(\text{Picture}) \lor \varepsilon(\text{Caption})) \Upsilon \tilde{P}(\text{MakeSave})))
\]

expressing the property that “After finishing the update, it is possible to add a picture or a caption until MakeSave follows as the next observable action.” This formula holds if $\text{Act}$ is given as $\{\text{Update}, \text{MakeSave}\}$. However, it does not hold when the action label $\text{Update}$ is added to $\text{Act}'$.

We are now ready to present the main result of this section, namely that the properties expressible in weak SE-LTL are preserved by the state/event stuttering equivalence.

Theorem 5.6 (Equivalence Preserves Properties). Let $\text{Act}'$ be a set of actions, $\text{Ap}'$ a set of atomic propositions, and let $\pi$ and $\sigma$ be two runs such that $\pi \equiv_{\text{Ap}'}^{\text{Act}'} \sigma$. Then for each formula $\varphi$ of wSE-LTL with respect to $\text{Act}'$ such that it only contains atomic propositions from $\text{Ap}'$, $\pi \models \varphi$ if and only if $\sigma \models \varphi$.

In order to prove this theorem, we need an auxiliary definition of a tail and a useful lemma. A tail of a run is its maximal subrun with a strictly smaller signature, i.e. a subrun that begins after first interesting action or first change in interesting state atomic propositions.

Definition 5.7 (Tail). Let $\pi = s_0, a_0, a_1, a_i, \ldots$ be a run of LKS $(S, \text{Act}, \Delta, S_{\text{Init}}, \text{Ap}, \mathcal{L})$, let $\text{Act}' \subseteq \text{Act}$ and $\text{Ap}' \subseteq \text{Ap}$. We define a tail of $\pi$ with respect to $\text{Act}'$ and $\text{Ap}'$, denoted as $\text{tail}_{\text{Ap}'}^{\text{Act}'}(\pi)$, as the maximal subrun $\pi^k$ of $\pi$ such that $\text{sig}_{\text{Ap}'}^{\text{Act}'}(\pi) = E_0, \alpha, \text{sig}_{\text{Ap}'}^{\text{Act}'}(\pi^k)$ for some $E_0 \subseteq \text{Ap}'$ and $\alpha \in \text{Act}' \cup \{\tau\}$. Note that the tail is undefined for runs whose signature is of the form $E_0$.

The following lemma asserts the existence of a function that provides mapping between equivalent runs preserving state/event stuttering equivalence of their subruns (see the illustration in Fig. 4).

Lemma 5.8 (Equivalence of Subruns). Let $\pi$ and $\sigma$ be two runs such that $\pi \equiv_{\text{Ap}'}^{\text{Act}'} \sigma$. Then there is a nondecreasing function $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ such that $f(0) = 0$ and for each $i \geq 0$, $\pi^i \equiv_{\text{Ap}'}^{\text{Act}'} \sigma^{f(i)}$. Moreover, for each $i$, the following holds:

\[
\sigma^{f(i)} \equiv_{\text{Act}'}^{\text{Ap}'(i)+1} \cdots \equiv_{\text{Act}'}^{\text{Ap}'(i+1)-1}
\]
Proof. We define the function \( f \) inductively as follows:

- \( f(0) = 0. \) Obviously \( \pi^0 = \pi \equiv_{\text{Act}'} \sigma = \sigma^0. \)
- Let \( f \) be defined for \( 0, \ldots, n \) and we want to define \( f(n+1). \) There are two cases:
  - \( \ell(\pi, n) \not\in \text{Act}' \) and \( \mathcal{L}(\pi(n)) \cap \text{Act}' = \mathcal{L}(\pi(n+1)) \cap \text{Act}'. \) Then \( f(n+1) = f(n). \) It is clear that \( \pi^{n+1} \equiv_{\text{Act}'} \pi^n \equiv_{\text{Act}'} \mathcal{L}(\pi(n+1)) \cap \text{Act}' \).
  - \( \ell(\pi, n) \in \text{Act}' \) or \( \mathcal{L}(\pi(n)) \cap \text{Act}' \not\subset \mathcal{L}(\pi(n+1)) \cap \text{Act}'. \) Then \( \pi^{n+1} = \text{tail}_{\text{Act}'}(\pi^n). \) We define \( f(n+1) \) such that \( \sigma^{f(n+1)} = \text{tail}_{\text{Act}'}(\sigma^{f(n)}). \) It is easily seen that if two runs are state/event stuttering equivalent, their tails also are, therefore \( \pi^{n+1} = \text{tail}_{\text{Act}'}(\pi^n) \equiv_{\text{Act}'} \text{tail}_{\text{Act}'}(\sigma^{f(n)}) = f^{f(n+1)} \).

It should be clear that the function \( f \) is nondecreasing. It remains to show that \( \sigma^{f(i)} \equiv_{\text{Act}'} \sigma^{f(i)+1} \equiv_{\text{Act}'} \cdots \equiv_{\text{Act}'} \sigma^{f(i)+i-1} \) for all \( i. \) If this condition were violated then both the signature of \( \sigma^{f(i)} \) and the signature of its tail would differ from the signature of \( \sigma^{f(i)+1}. \) Clearly, from the construction of \( f \) this is not possible. \( \square \)

We can proceed to the proof of Theorem 5.6.

Proof of Theorem 5.6. Let \( \text{sig}_{\text{Act}'}(\pi) = \text{sig}_{\text{Act}'}(\sigma) \). The proof follows by induction on the formula. We only show three interesting cases and only one direction, i.e., that \( \pi \models \varphi \) implies \( \sigma \models \varphi. \) The other cases are evident and the other direction follows from the symmetry of the equivalence.

- \( \pi \models \psi_1 \ U \ \psi_2. \) Then there is some \( k \) such that \( \pi^k \models \psi_2 \) and for each \( j < k, \pi^j \models \psi_1. \) But then Lemma 5.8 together with the induction hypothesis implies that \( \sigma^{f(k)} \models \psi_2 \) and for each \( i < f(k), \sigma^i \models \psi_1, \) that is \( \sigma \models \psi_1 \ U \ \psi_2. \)
- \( \pi \models \tilde{X} \varphi. \) Then there is some \( k \) such that \( \ell(\pi, k) \) is the first interesting action on \( \pi \) and \( \pi^{k+1} \models \varphi. \) From the state/event stuttering equivalence, there has to be some \( j \) such that \( \ell(\sigma, j) \) is the first interesting action on \( \sigma \) and \( \text{sig}_{\text{Act}'}(\sigma^{f(j)}) = \text{sig}_{\text{Act}'}(\pi^{k+1}) \) is the part of the original signature that starts after the first interesting action. Thus \( \sigma \models \tilde{X} \varphi. \)
- \( \pi \models \psi_1 \ U a \ \psi_2. \) Then there is some \( k \) such that \( \ell(\pi, k) = a, \pi^{k+1} \models \psi_2 \) and for all \( j < k + 1, \pi^j \models \psi_1. \) As in the case of \( \tilde{X} \varphi, \) it follows from Lemma 5.8 that \( \sigma^{f(k+1)} \models \psi_2 \) and \( \sigma^i \models \psi_1 \) for all \( i < f(k+1). \) We need to show that \( \ell(\sigma, f(k+1) - 1) = a. \) However, we know that \( \text{sig}_{\text{Act}'}(\sigma^k)) = E, a, \text{sig}_{\text{Act}'}(\pi^{k+1}) \) for some \( E \) and we know from Lemma 5.8 that \( \sigma^{f(k+1)-1} \equiv_{\text{Act}'} \sigma^{f(k)} \equiv_{\text{Act}'} \pi^k. \) Therefore, \( \text{sig}_{\text{Act}'}(\sigma^{f(k+1)-1})) = E, a, \text{sig}_{\text{Act}'}(\sigma^{f(k)})) \) and thus \( \ell(\sigma, f(k+1) - 1) = a \) and \( \sigma \models \psi_1 \ U a \ \psi_2. \) \( \square \)

What remains is to show that wSE-LTL is indeed a weak version of SE-LTL, i.e., that all properties expressible in wSE-LTL are also expressible in SE-LTL. The following theorem states that every formula of wSE-LTL can be translated in linear time to an equivalent formula of SE-LTL. This way the verification problem for wSE-LTL can be reduced to the verification problem for SE-LTL, which is solvable in a way similar to the standard LTL verification as described in [1].

Theorem 5.9 (Embedding of wSE-LTL into SE-LTL). Every formula \( \varphi \) of weak SE-LTL with respect to Act’ can be translated to a formula \( T(\varphi) \) of SE-LTL such that for each \( \pi, \pi \models \varphi \) if and only if \( \pi \models T(\varphi) \).

Proof. We define an auxiliary formula \( \xi := \bigwedge_{a \in \text{Act}'} \neg \text{P} (a) \). The translation is defined inductively as follows:

\[ T(p) := p \]
\[ T(\tilde{\varphi}(a)) := \xi \ U \ \text{P} (a) \]
\[ T(\tilde{X} \varphi) := \xi \ U (\neg \xi \land X T(\varphi)) \]
\[ T(\varphi \ U \psi) := T(\varphi) \ U T(\psi) \]
\[ T(\varphi \ U a \ \psi) := T(\varphi) \ U (\text{P} (a) \land T(\varphi) \land X T(\psi)) \]
\[ T(\varphi \ U \psi) := T(\varphi) \ U T(\psi) \]
\[ T(\neg \varphi) := \neg T(\varphi) \]
The correctness of the construction is proved by induction. The interesting cases are $\bar{\tau}$, $\bar{X}$ and $U_\nu$.

$\pi \models \bar{\tau}(a) \iff \exists k \geq 0 : \ell(\pi, k) = a$ and $\forall j < k : \ell(\pi, j) \not\in Act'$

$\iff \exists k \geq 0 : k \models T(\psi)$ and $\forall j < k : \pi^j \models T(\psi)$

$\iff \pi \models \bar{X} \psi \iff \exists k \geq 0 : \ell(\pi, k) \in Act', \forall j < k : \ell(\pi, j) \not\in Act'$ and $\pi^{k+1} \models \psi$

$\iff \exists k \geq 0 : \pi^k \models \bar{\tau}(\psi)$ and $\forall j < k : \pi^j \models T(\psi)$

The remaining cases are straightforward. □

### 5.1. Characterization of invariant SE-LTL properties

We have shown that wSE-LTL is preserved by state/event stuttering equivalence and can be embedded into SE-LTL. Thus, wSE-LTL can be seen as a characterization of some SE-LTL properties that are preserved by state/event stuttering equivalence (we use the term state/event stutter-invariant for such properties in the following). We now show that this characterization is exact, i.e. that all state/event stutter-invariant SE-LTL properties are expressible in wSE-LTL. The proof follows the method of [10].

**Definition 5.10.** A run $\pi$ is state/event stutter-free, if for each $i \geq 0$ one of the following holds:

- $\ell(\pi, i) \in Act'$ (ith transition is labelled by interesting action)
- $L(\pi(i)) \cap Ap' \neq L(\pi(i + 1)) \cap Ap'$ (ith transition changes the state labelling)
- $\ell(\pi, j) \not\in Act'$ and $L(\pi(j)) \cap Ap' = L(\pi(j + 1)) \cap Ap'$ for all $j \geq i$ (nothing interesting ever happens from ith position onwards).

It is clear that a state/event stutter-free run is a unique representant of its state/event stuttering equivalence class. Note that an arbitrary subrun of a state/event stutter-free run is also state/event stutter-free.

**Theorem 5.11.** Every state/event stutter-invariant property expressible in SE-LTL is expressible in wSE-LTL.

We will show that for every SE-LTL formula $\varphi$ there exists a wSE-LTL formula $\tau(\varphi)$ that agrees with $\varphi$ on all state/event stutter-free runs. Clearly this implies the theorem.

The formula $\tau(\varphi)$ is defined inductively as follows. The straightforward parts are $\tau(p) = p$ for $p \in Ap$, $\tau(\neg \varphi) = \neg \tau(\varphi)$, $\tau(\varphi \land \psi) = \tau(\varphi) \land \tau(\psi)$ and $\tau(U_\psi) = \tau(\psi) U_\psi \tau(\psi)$. These choices are obviously correct. The more difficult parts are that of $T(\varphi)$ and $X \varphi$.

Assume that $Ap = \{p_1, \ldots, p_n\}$ and let $N$ be the set of all subsets of $Ap$, i.e. $N = 2^{Ap}$. For each $v \in N$, let $\beta_v$ be the formula $\alpha_1 \land \cdots \land \alpha_n$ where $\alpha_j = p_j$ if $p_j \in v$ and $\alpha_j = \neg p_j$ otherwise. If $Ap = \emptyset$ then $N = \{\emptyset\}$ and $\beta_v = \text{true}$. Thus, $\beta_v$ holds in precisely those states whose valuation is equal to $v$.

The two remaining formulae can then be dealt with as follows:

\[
\tau(T(\varphi)) := \exists v \in N \beta_v U_v \text{true}
\]

\[
\tau(X \varphi) := \psi_1 \lor \psi_2 \lor \psi_3,
\]

where

\[
\psi_1 := \bigvee_{v \in N} (G \beta_v \land \neg \bar{X} \text{true} \land T(\varphi))
\]

\[
\psi_2 := \bigvee_{v \in N \in Act'} \big( \beta_v U_v \text{true} \land \bar{X} \tau(\varphi) \big)
\]

\[
\psi_3 := \neg \big( \bigvee_{v \in N \in Act'} (\beta_v U_v \text{true}) \lor \bigvee_{v' \in N \setminus \{v\}} (\beta_v \land (\beta_{v'} U_{v'} \tau(\varphi))) \big)
\]
In order to prove the correctness of this choice, we first prove the following lemma.

**Lemma 5.12.** Let \( \pi \) be a state/event stutter-free run. Then \( \ell(\pi, 0) \in \Act \) if and only if \( \pi = e \in \Act \).

**Proof.** If \( \pi = s_0, a, \ldots \) then clearly \( \pi = \beta(a, \true) \) where \( a \) is the set of all state propositions true at \( s_0 \).

The other direction is proved by contradiction. Suppose that \( \ell(\pi, 0) \notin \Act \) and that \( \pi \) satisfies the formula in the lemma. Then, there is some \( a \) such that \( \pi = \beta(a, \true) \). Therefore, \( \text{pr}_\Act(\pi) \) has to be of the form \( E_0, \tau, E_1, \ldots \), where both \( E_0 \) and \( E_1 \) are equal to \( a \). But then, \( \pi \) is not state/event stutter-free, which is a contradiction. \( \square \)

We may now prove the correctness of \( \tau(\mathcal{P}(a)) \) and \( \tau(X \varphi) \). The choice of \( \tau(\mathcal{P}(a)) \) is evidently correct by an argument similar to that of Lemma 5.12.

Let us now suppose that \( \pi \models X \varphi \). We want to show that then \( \pi \models \tau(X \varphi) \). It follows from the induction hypothesis that \( \pi^1 \models \tau(\varphi) \). As \( \pi \) is state/event stutter-free run, it has to be of one of the following three forms:

(i) \( \text{pr}_\Act(\pi) = E_0, \tau, E_1, \tau, E_2, \tau, \ldots \), where \( E_0 = E_1 = E_2 = \ldots \), i.e. nothing interesting ever happens on \( \pi \). In this case it is obvious that \( \pi \models G \beta(v) \) where \( v = E_0 \) and \( \pi \models \neg \widetilde{\varphi} \). As \( \text{pr}_\Act(\pi) = \text{pr}_\Act(\pi^1) \), we also know that \( \pi \models \tau(\varphi) \). Thus \( \pi \models \psi_1 \) and therefore \( \pi \models \tau(X \varphi) \).

(ii) \( \text{pr}_\Act(\pi) = E_0, a, E_1, \ldots \), where \( a \in \Act \), i.e. first transition of \( \pi \) has interesting label. Then, it follows from Lemma 5.12 that \( \pi \models \beta(a, \true) \) for some \( a \) and \( v \). From the definition of \( \widetilde{\varphi} \) and the fact that \( \pi^1 \models \tau(\varphi) \) it then follows that \( \pi \models \widetilde{\varphi} \). Thus \( \pi \models \psi_2 \) and therefore \( \pi \models \tau(X \varphi) \).

(iii) \( \text{pr}_\Act(\pi) = E_0, \tau, E_1, \ldots \), where \( E_0 \neq E_1 \), i.e. first transition of \( \pi \) has noninteresting label but changes state labelling.

In this case it is clearly seen that \( \pi \models \beta(v) \land (\beta(v') \land \tau(\varphi)) \), where \( v = E_0 \) and \( v' = E_1 \). That \( \pi \models \neg \beta(v, \true) \) for every value of \( v \) and \( \pi \) follows from Lemma 5.12. Thus \( \pi \models \psi_3 \) and therefore \( \pi \models \tau(X \varphi) \).

For the other direction of the proof, suppose that \( \pi \models \tau(X \varphi) \). We want to show that \( \pi \models X \varphi \). The induction hypothesis is that whenever a state/event stutter-free run satisfies \( \tau(\varphi) \), it also satisfies \( \varphi \). There are three cases:

(a) \( \pi \models \psi_1 \). As \( \pi \models G \beta(v) \) and \( \pi \models \neg \widetilde{\varphi} \), the run \( \pi \) has to be of the form (i). As \( \pi \models \tau(\varphi) \) and \( \pi \models \pi^1 \models \varphi \) and thus \( \pi \models X \varphi \).

(b) \( \pi \models \psi_2 \). By Lemma 5.12, \( \pi \) has to be of the form (ii), i.e. its first transition has to be labelled by an interesting action. But then \( \pi \models \widetilde{\varphi} \). Thus \( \pi \models \psi_2 \) and therefore \( \pi \models X \varphi \).

(c) \( \pi \models \psi_3 \). By Lemma 5.12, \( \pi \) has to be of either the form (i) or (iii). As \( \pi \models \beta(v') \land (\beta(v') \land \tau(\varphi)) \) for some \( v \) and \( v' \) such that \( v \neq v' \), it cannot be the form of (i). Thus, it has to be that \( \text{pr}_\Act(\pi) = E_0, \tau, E_1, \ldots \), where \( E_0 \neq E_1 \). Then clearly \( v = E_0 \) and \( \pi \models \beta(v') \land \tau(\varphi) \) as it cannot be the case that \( \pi \models \beta(v) \) because \( \pi \) is state/event stutter-free. But then \( \pi \models \psi_3 \) and \( \pi \models X \varphi \). This completes the proof of Theorem 5.11.

Although the construction in the above proof is exponential in the worst case, the main significance of this result is that using weak SE-LTL comes without loss of expressiveness over SE-LTL if we want to use state/event stutter-invariant properties. Moreover, the construction justifies the choice of \( \mathcal{P}, \varphi \) and \( \true \) operators, which made the construction possible. Note that if any of these were excluded from the logic, it would be less expressive and the above result would not hold.

### 6. Partial order reduction

The goal of this section is to show that the partial order reduction method can be applied to LKSs so that the reduced LKS remains state/event stuttering equivalent. At first, we summarize the basics of the partial order reduction method. While presenting the method we follow the explication from [16]. Consequently, we explain how the method can be applied to SE-LTL.

**Definition 6.1 (State Transition System).** A state transition system is a triple \((S, T, s_{init})\) where \( S \) is a set of states, \( s_{init} \) is an initial state and \( T \) is a set of transitions such that for each \( \alpha \in T \), \( \alpha \subseteq S \times S \). Further, for each \( \alpha \in T \) and for each state \( s \in S \) there is at most one \( s' \in S \) such that \( (s, s') \in \alpha \).

An initial transition path of a state transition system is an infinite sequence \( \alpha_0 \alpha_1 \ldots \) such that there are states \( s_0, s_1, \ldots \) satisfying \( s_0 = s_{init} \) and for all \( i \), \( (s_i, s_{i+1}) \in \alpha_i \).

The idea of the ample set method [8,17] is to construct a reduced state space by choosing a smaller set of successors at each state. Instead of exploring all successors from a given state, denoted as \( \text{enabled}(s) \), we explore only successors from \( \text{ample}(s) \subseteq \text{enabled}(s) \).

**Theorem 6.2 ([16]).** Let \( M \) be a state transition system and let \( V \subseteq T \) be an arbitrary set of visible transitions. Let \( M' \) be the reduced system constructed using the ample set partial order algorithm. Then for each initial transition path \( \pi \) from \( M \) there is an initial transition path \( \pi' \) in \( M' \) such that \( \pi \) and \( \pi' \) have the same sequence of visible transitions.

**Proof.** The theorem follows from the proof of Theorem 12 in [16]. \( \square \)
Now we present a transformation of an LKS to a state transition system. From now on, we fix the LKS $\mathcal{K} = (S, \text{Act}, \Delta, s_{\text{init}}, \mathcal{P}, \mathcal{L})$, and two sets, $\text{Act}' \subseteq \text{Act}$ and $\mathcal{P}' \subseteq \mathcal{P}$ of interesting actions and interesting atomic propositions respectively. We need the notions of invisibility and proper transition partition first.

**Definition 6.3 (Invisibility).** A transition $(q, a, r) \in \Delta$ is called invisible if $a \notin \text{Act}'$ and $\mathcal{L}(q) \cap \mathcal{P}' = \mathcal{L}(r) \cap \mathcal{P}'$. A transition is called visible if it is not invisible.

**Definition 6.4 (Proper Transition Partition).** An indexed set $P = \{ \Delta_i \mid i \in I \}$ is called a proper transition partition if the following holds:

- $P$ is a partition of $\Delta$, i.e. $\bigcup_{i \in I} \Delta_i = \Delta$ and $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$.
- $P$ preserves actions, i.e. $\forall i \in I. \exists a \in \text{Act} : \Delta_i \subseteq S \times \{a\} \times S$.
- $P$ is deterministic, i.e. for all $i$ and for all $s \in S$, there is at most one $s' \in S$ such that $(s, a, s') \in \Delta_i$.

We now present a transformation of an LKS $\mathcal{K} = (S, \text{Act}, \Delta, s_{\text{init}}, \mathcal{P}, \mathcal{L})$ with a proper transition partition $P$ into a state transition system $\mathcal{M} = (S, T, s_{\text{init}})$. Let $T = \{ \alpha_i \mid i \in I \}$ and $\alpha_i = \{(s, a, s') \in \Delta_i \mid a \in \text{Act} \}$. The set of visible transitions $V \subseteq T$ is defined as $V = \{ \alpha_i \mid \text{at least one transition in } \Delta_i \text{ is visible} \}$. We denote $\Delta(\alpha_i) = \Delta_i$ the underlying partition set for $\alpha_i$. Let $\alpha = \alpha_0\alpha_1\ldots$ be an initial transition path of the state transition system $(S, T, s_{\text{init}})$. We assign to $\alpha$ an initial run $s_0, a_0, s_1, a_1, \ldots$ where $s_0 = s_{\text{init}}, (s_i, a_i, s_{i+1}) \in \Delta(\alpha_i)$ for $i \geq 0$. Clearly, the assigned initial run is a unique initial run of the LKS due to the nature of the proper transition partition.

**Theorem 6.5.** Let $\alpha = \alpha_0\alpha_1\ldots$ and $\beta = \beta_0\beta_1\ldots$ be two initial transition paths of the state transition system $(S, T, s_{\text{init}})$. Let $\pi$ and $\sigma$ be the initial runs assigned to $\alpha$ and $\beta$, respectively. If $\alpha$ and $\beta$ have the same sequence of visible transitions, then $\pi \equiv_{\text{Act}}^\mathcal{P} \sigma$.

**Proof.** Invisible transitions of the state transition system represent transitions of the original LKS such that they do not contribute to the runs’ signatures. 

**Corollary 6.6.** Let $M$ be an LKS and let $\text{Act}' \subseteq \text{Act}$ and $\mathcal{P}' \subseteq \mathcal{P}$. Let $M'$ be the reduced system constructed using the ample set partial order algorithm on the state transition system that is created as discussed above. Then for each initial run $\pi$ of $M$ there is an initial run $\sigma$ of $M'$ such that $\pi \equiv_{\text{Act}}^\mathcal{P} \sigma$.

Three things have to be supplied to the partial order reduction algorithm along with the LKS to be verified in order for the method to work. They are the proper transition partition, the set of interesting atomic propositions $\mathcal{P}'$ and the set of interesting actions $\text{Act}'$.

A proper transition partition can be constructed automatically from some additional information of the structure of the LKS in question. Take for instance that the LKS represents a component-based system made from a number of smaller components such as the LKS in Example 3.2. A proper transition partition can be constructed in such a way that all transitions of the system that correspond with one transition of a smaller component constitute exactly one set of the partition.

A set of interesting atomic propositions is acquired from the verified formula. It is constructed as the set of all atomic propositions present in the formula. A set of interesting actions, however, has to be supplied by the user by hand. This set has to be a part of the property specification, as noted in Section 5, nonetheless. The cardinality of this set can affect the efficiency of the reduction method. It is thus desirable to specify as small set of interesting actions as possible, bearing in mind the intended semantics of the verified formula.

The partial order reduction itself can then be done on the fly during the automata-based verification process. Known methods and implementations can be used for this purpose, see e.g. [18].

**Example 6.7.** Consider the LKS from Example 3.2. If we choose $\mathcal{P}'$ to be an arbitrary subset of $\mathcal{P}$ and $\text{Act}'$ to be an arbitrary subset of $\text{Act}$ such that $\mathcal{MakeLog} \notin \text{Act}'$ then the state space of the reduced LKS will look as depicted in Fig. 5. In this case $\text{ample}(s) = \text{enabled}(s)$ for all states except $H$ and $\text{ample}(H) = \{I\}$.

### 7. Implementation and experimental results

The previous sections present a general method of applying the partial order reduction method to verification of state/event LTL properties on labelled Kripke structures. In this section, we describe an application of this method on a concrete modelling formalism, the component-interaction automata [3,5]. Component-interaction automata were devised to model and verify interactions among components in component-based systems. SE-LTL is suitable for expressing interaction behaviour of component-based systems, as it is able to capture both actual and possible (enabled) interaction of components, as well as their combinations. Due to the hierarchical structure of the component-interaction automata models, new heuristics for approximating the ample sets in partial order reduction are needed.

We provide a short description of the formalism, make a few remarks about the implementation process and conclude with some results obtained by using the method on a case study.

**Definition 7.1 (CI Automaton [3]).** A component-interaction automaton (CI automaton for short) is a 5-tuple $C = (Q, A, \delta, I, H)$ where $Q$ is a finite set of states, $A$ is a finite set of actions, $\Sigma = \{(S_H \cup \{\} ) \times A \times (S_H \cup \{\} )\} \setminus \{\{\} \times A \times \{\} \}$ is a set of labels, $\delta \subseteq Q \times \Sigma \times Q$ is a finite set of labelled transitions, $I \subseteq Q$ is a nonempty set of initial states, and $H$ is a structured tuple representing a hierarchy of component names where the set of component names is denoted $S_H$. 


The labels have semantics of input, output, or internal, based on their structure. In the triple, the middle item represents an action name, the first item represents a name of the component that outputs the action, and the third item represents a name of the component that inputs the action. At most one of the component names can be replaced with the special symbol "−", meaning that the environment is supposed to send/receive the action or that the communication partner is to be specified later in a composition.

To compose components into a higher-level component a composition operator is defined. Automata can be composed together using a parametrizable composition operator $\otimes^F$, which composes a given finite set of automata with respect to the set of feasible labels $F$. Given a set of labels $F$, the operator composes the set of CI automata into a product automaton allowing only those transitions from the product that have labels from $F$. In the product, the components cooperate either by interleaving of their original transitions, or by simultaneous execution of two complementary transitions (with labels $(n_1, a, −), (−, a, n_2)$) which results in a new internal transition (with label $(n_1, a, n_2)$).

In our verification setting, the system to be verified consists of a finite set of simple CI automata, whose state space is explicitly described. The simple automata are composed in a hierarchical way, forming several composite automata. The automaton which is highest in the composition hierarchy then stands for the system that is to be verified.

This automaton can be seen as a LKS where the set of actions of the LKS corresponds to the set of labels of the CI automaton, the set of atomic propositions is defined as $Ap = \{E(m, a, n) \mid (m, a, n) \in \Sigma\}$, and the state-labelling function is defined as $\mathcal{L}(s) = \{E(m, a, n) \mid transition with label (m, a, n) is enabled in s\}$.

To construct a proper transition partition (Definition 6.4), and thus a state transition system out of this LKS, we exploit the structure of the system. We construct the partition so that we put together those transitions that involve the same transitions of the simple automata. This way, the transitions of the induced state transition system are of two kinds: those that represent an execution of just one of the simple automata, and those that represent a synchronization between two simple automata.

Now we are ready to employ the ample set algorithm from [16]. The ample set conditions are the following.

- **C0**: $ample(s) = \emptyset$ if and only if $enabled(s) = \emptyset$.
- **C1**: Along every path in the full state graph that starts at $s$, a transition that is dependent on a transition in $ample(s)$ cannot be executed without a transition from $ample(s)$ occurring first.
- **C2**: If $enabled(s) \neq ample(s)$ then every $\alpha \in ample(s)$ is invisible.
- **C3**: A cycle is not allowed if it contains a state in which some transition $\alpha$ is enabled, but is never included in $ample(s)$ for any state $s$ on the cycle.

As checking these conditions is hard in general, we overapproximate them, using the heuristics from [16]. However, in our specific setting, the heuristics have to be modified in two points. First, we change the strategy of choosing the candidate sets. Instead of choosing just the enabled transitions of a single simple automaton, we try to choose enabled transitions of the composite automata. Second, we use a different heuristic for checking the condition C1. The reason for this is that the
heuristics described in [16] are not well suited for a setting with many synchronization transitions, and almost all transitions in a composite CI automaton are expected to be synchronizations. We therefore propose the following condition:

$$\forall \alpha \in \text{current}(s) : \alpha \text{ is an internal transition of automaton } i$$

where $i$ is the automaton whose enabled transitions were chosen as the candidate set, $\text{current}(s)$ are all transitions that belong to automaton $i$ and are either enabled in state $s$ or in a different state $s'$ with the property that the states of the simple automata $s_j$ such that $j$ is included in (possibly composite) automaton $i$ coincide with $s'_j$. This condition can be easily verified directly from the description of the simple automata. A more detailed explanation of the heuristics and implementation details was given in [19].

To provide some evidence for the effectiveness of the partial order method, we apply our implementation on a case study. The model we choose was created as a part of the CoCoME component modelling contest [5]. Our previous experience with verification of this model, which has uncovered the need of a partial order reduction method for state/event systems, is reported upon in [6].

The modelled system, the Trading System (see Fig. 6), serves to handle sales in a chain of supermarkets. Its functionality includes the interaction with the cashier at the cash desk, like product scanning, price lookup, cash/card payment, and bill printing, as well as accounting the sale at the inventory, or determining whether an express cash desk is needed in the store. Furthermore, the Trading System deals with ordering goods from wholesalers, and generating various kinds of reports. The system is open, designed to interact with external components representing users of the system (cashiers and managers) and a bank application. The model of the system consists of 140 simple CI automata, composed hierarchically into 34 composite automata up to 6 levels of depth. The behaviour of the model features a high degree of independent interleaving of actions, it can thus be expected to achieve a fair amount of state space reduction using the partial order reduction method.

The results obtained by using the method are summarized in Table 1. The various models the verification was performed on were created by complementing the Trading Systems with various components depicting the users of the system. Here, C stands for one cashier with complex behaviour, which is the user component that has been used for verification in [6]; SC stands for one cashier with simple behaviour, it supplies just one action to the system; SCM stands for simple cashier operating concurrently with complex manager; SCR stands for simple cashier performing its only action repeatedly in a cycle; SCSM stands for simple cashier with simple manager; and TSC stands for two simple cashiers using the system concurrently. For each of these variants, three properties were verified, those corresponding with properties 2, 5 and 9 as described in [6]. The table shows the number of states of each model combined with each property and the memory and time that was needed to generate the state space, both with and without employing the partial order reduction. A dash (—) in the table means that the information is not available, as the process of state space generation exceeded the maximum of 4 GB of memory.

Our experience with applying the partial order reduction on this case study is very positive. In all cases, the state space has been reduced to at least one-tenth of the original size and there have been cases where the reduction ratio surpassed one hundred.
Table 1
Experimental results (the reduction ratio relates the number of states of the model to the number of states obtained after applying partial order reduction).

<table>
<thead>
<tr>
<th>Model</th>
<th>Without p.o.r.</th>
<th>With p.o.r.</th>
<th>Reduction ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># states</td>
<td>RAM (MB)</td>
<td>Time (s)</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>749340</td>
<td>139</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>1498679</td>
<td>274</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>750684</td>
<td>139</td>
</tr>
<tr>
<td>SC</td>
<td>2</td>
<td>29341</td>
<td>9</td>
</tr>
<tr>
<td>SC</td>
<td>5</td>
<td>58681</td>
<td>15</td>
</tr>
<tr>
<td>SC</td>
<td>9</td>
<td>29629</td>
<td>9</td>
</tr>
<tr>
<td>SCM</td>
<td>2</td>
<td>22745391</td>
<td>4045</td>
</tr>
<tr>
<td>SCM</td>
<td>5</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>SCM</td>
<td>9</td>
<td>22915023</td>
<td>4076</td>
</tr>
<tr>
<td>SCR</td>
<td>2</td>
<td>2994016</td>
<td>570</td>
</tr>
<tr>
<td>SCR</td>
<td>5</td>
<td>5988032</td>
<td>1128</td>
</tr>
<tr>
<td>SCR</td>
<td>9</td>
<td>3034336</td>
<td>578</td>
</tr>
<tr>
<td>SCRM</td>
<td>2</td>
<td>6369598</td>
<td>1135</td>
</tr>
<tr>
<td>SCRM</td>
<td>5</td>
<td>12739195</td>
<td>2263</td>
</tr>
<tr>
<td>SCRM</td>
<td>9</td>
<td>6413518</td>
<td>1143</td>
</tr>
<tr>
<td>TSC</td>
<td>2</td>
<td>1356277</td>
<td>245</td>
</tr>
<tr>
<td>TSC</td>
<td>5</td>
<td>2712553</td>
<td>484</td>
</tr>
<tr>
<td>TSC</td>
<td>9</td>
<td>1373653</td>
<td>248</td>
</tr>
</tbody>
</table>

8. Conclusion and future work

The paper introduces a partial order reduction technique for state/event LTL. The technique is based on a new stuttering equivalence, which is able to reflect both state and transition labels while regarding both with a different principle to closely fit their nature. On the level of states, the stuttering concentrates on changes in assigned atomic propositions along a run, whereas in the case of actions, the interesting events are observed at every single occurrence of an action representing interesting behaviour of the system, which is stated explicitly with respect to the verified property for instance. The paper moreover gives the characterization of the state/event LTL properties preserved by the equivalence, and summarizes the attributes into the definition of its fragment, called weak state/event LTL. This fragment is preserved by the equivalence while staying strong enough to reflect interesting component-specific properties, discussed at the beginning of this paper.

After introducing both the equivalence and the corresponding logic fragment, we discuss the partial order reduction technique. We show that the partial order reduction task for the state/event case can be translated into the state-based case via providing existing algorithms with a modified definition of transition invisibility reflecting the discussed specifics. The advantage of such translation is that known algorithms may be used for solving the problem. Moreover, the efficiency of the partial order reduction method, which has been experimentally verified on a number of case studies [18,20,21], can be attributed to our approach as well.

Finally, after presenting the partial order method for state/event LTL in theory, we shortly describe its application on the formalism of component-interaction automata. We also present some of the results obtained by employing the partial order method within our verification tool [22,23]. The experimental study demonstrates the efficiency of the method which can push the size boundary within which component-based systems can be effectively model checked.

As for the future work, our general aim is in exploring reduction methods for verification of component-based systems. We plan both to investigate the applicability of existing approaches (as we did in this paper) and to discover new approaches based on the structure of such systems. Concerning the partial order reduction method, we plan to perform more experiments on more case studies as well as to look at possible extensions to the logic, e.g. fluent LTL [24]. We also plan to combine the partial order reduction with parallel and multi-core computing by incorporating our verification tool into the DiVinE 2.0 framework [25].

Acknowledgements

We acknowledge our colleagues, Milan Křivánek and Filip Štefaňák, for their help with implementation and experimental study.

References


