Unfolding Concurrent Well-Structured Transition Systems*

Frédéric Herbreteau, Grégoire Sutre, and The Quang Tran

LaBRI, CNRS UMR 5800, Domaine Universitaire, Talence, France
{fh, sutre, tran}@labri.fr

Abstract. Our main objective is to combine partial-order methods with verification techniques for infinite-state systems in order to obtain efficient verification algorithms for concurrent infinite-state systems. Partial-order methods are commonly used in the analysis of finite systems consisting of many parallel components. In this paper we propose an extension of these methods to parallel compositions of infinite-state systems. We argue that it is advantageous to model each component by an event structure as this allows us to exhibit the concurrency present implicitly in some infinite-state systems such as automata with queues or counters. We generalize the notion of complete prefix from 1-safe Petri nets to all well-structured transition systems. We give an on-the-fly unfolding algorithm which given event structures representing the components produces an event structure representing their synchronized product. A prototype implementation demonstrates the benefits of our approach.

1 Introduction

Partial-order methods [God96, Val91, Pel93] are frequently used for the verification of programs, in particular for systems of concurrent processes. Indeed, proving that the computations of such systems meet some requirement often results in the well-known exponential blow-up due to interleaving of concurrent actions. Partial-order methods tackle this problem by selecting a hopefully small set of relevant computations that are sufficient to conclude. Selecting among the interleavings is sound because ordering concurrent (independent) actions is irrelevant. Hence, instead of considering totally ordered computations, one analyses *traces* [Maz86] that stand for all equivalent computations w.r.t. concurrency. As a consequence, the whole computation tree can be partially rather than totally ordered resulting in *event structures* [NPW81, NRT95]. Efficient exploration techniques, e.g. unfolding [McM95, NRT95], exist for partially-ordered structures.

Traditionally, partial-order methods have been applied to concurrent *finite-state* processes and proved to be successful. In this paper, we apply partial-order methods to parallel compositions of *infinite-state* systems such as counter machines and communicating finite-state machines. Verification of infinite-state systems is known to be undecidable, however some classes of infinite-state systems enjoy nice decidability results.

* This work was partially supported by the French Ministry of Research (Project PERSÉE of the ACI Sécurité et Informatique).

Improved and extended (unofficial) version of the paper published in:
Proc. 13th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07), Braga, Portugal, Mar.–Apr. 2007, volume 4424 of Lecture Notes in Computer Science, pages 706–720. Springer, 2007.

Well-Structured Transition Systems [Fin90, AČJT00, FS01] provide us with a nice framework based on weak simulation relations that are well-preorders. Since a well-preorder admits no infinite decreasing sequence, one can consider only a finite prefix of the computation tree in order to decide properties like boundedness, termination or covering.

Our contribution. Our goal is to analyse synchronized products of well-structured systems using a method similar to Petri net unfolding in order to obtain event structures. The most straightforward approach would be to consider that each component generates an (infinite) transition system, then use an on-the-fly variant of the unfolding method for parallel composition of finite automata. This turns out not to be satisfactory: imagine that one models a counter by an (infinite) automaton then if, say, three different processes want to increase the counter, their actions will get interleaved on the automaton modeling the counter. As in principle those actions are independent, we lose a good deal of concurrency present in the original system. Our solution is to model each component by an event structure, hence taking advantage of the intrinsic concurrency present in each component.

Our first contribution is an algorithm for constructing event structures for components modeling counters. It results in event structures with more concurrency than in [McM95]. We also present a general algorithm that works for all (infinite) transition systems. This is not trivial: one wants the most concurrent event structure whereas concurrency is not explicit in transition systems. Of course, our algorithm is less efficient than dedicated algorithms (e.g. for counter automata), however it exhibits a good deal of concurrency using local independence. For instance it produces the same event structures as those reported in [LI05] for queue automata.

Our second contribution is a generalization of the unfolding method of [ER99] to parallel composition of potentially infinite event structures. Our algorithm is both capable of exploiting concurrency in components as well as among them. In particular we show that modeling Petri nets as the parallel composition of its places (viewed as counters) results in very efficient analysis using our algorithms.

Of course there is no hope to have a notion of complete prefix for a parallel composition of infinite systems. There is hope though when the components are well-structured systems. We give a property-preserving truncation criterion for event structures of well-structured transition systems. The resulting (complete) prefix contains enough information to decide boundedness, termination and quasi-liveness. We also show preservation of well-structure under parallel composition for all variants of well-structure. Remark that these results cannot be directly obtained from previous techniques on well-structured systems, since the complete prefix is not a compact partial-order representation of the finite reachability tree of [FS01].

Related work. The unfolding technique [McM95] has been developed for several models of concurrency, e.g. synchronous products of transition systems [ER99], high-level Petri nets [KK03], extended finite state machines [LI05], symmetric Petri nets [CGP01]. However, all these techniques deal with finite-state models.

In [AJKP98], the authors address the coverability problem for infinite state systems by combining partial-order reductions and symbolic backward computations. The unfolding of unbounded Petri nets was recently considered. In [AIN00] Abdulla et al. propose a backward unfolding technique for coverability analysis, and [DJN04] presents an unfolding-based adaptation of Karp and Miller's algorithm. Our method generalizes these results: it analyses any (infinite) well-structured transition system [FS01], offering both forward and backward approaches, hence enabling to check covering, boundedness and termination properties.

Outline. Section 2 introduces notations and definitions for transition systems and event structures. In section 3 we prove well-structure properties for event structures. Then, in section 4 we introduce our algorithms for unfolding systems. Finally, in section 5 we give some experimental results showing the benefits of our approach, and we conclude on future work. Please note that some preliminary (standard) definitions along with all proofs are given in appendix. This paper is the long version of our TACAS 2007 paper.

2 Labeled Transition Systems and Event Structures

A *binary relation* R on some set U is any subset of $U \times U$. We will sometimes view functions as relations. Given a subset $X \subseteq U$, we denote by $R[X]$ the *relational image* of X through R, defined by $R[X] = \{y \in U \,/\, \exists\, x \in X, x\, R\, y\}$. The *inverse* of R is the binary relation R^{-1} on U defined by $x\, R^{-1}\, x'$ iff $x'\, R\, x$. A *preorder* on some set U is any reflexive and transitive relation $\preceq$ on U. We let $x \prec x'$ denote $x \preceq x' \not\preceq x$. Given a preorder $\preceq$ on U, the *inverse* relation $\preceq^{-1}$ is a preorder also written $\succeq$. For any subset $X \subseteq U$, the set $\preceq[X]$ (resp. $\succeq[X]$) is called the *upward closure* (resp. *downward closure*) of X with respect to $\preceq$. We say that X is *upward-closed* (resp. *downward-closed*) if X is equal to its upward closure (resp. downward closure). A *partial order* on U is any antisymmetric preorder on U. Given a partial order $\leq$ on U, a *maximal element* (resp. *minimal element*) of some subset $X \subseteq U$ is any $m \in X$ such that $m' \not\geq m$ (resp. $m' \not\leq m$) for all $m' \neq m$ in X. We write $\mathsf{Max}_{\leq}(X)$ (resp. $\mathsf{Min}_{\leq}(X)$) for the set of maximal elements (resp. minimal elements) of X with respect to $\leq$.

Given a set Σ, we denote by Σ^* (resp. Σ^ω) the set of all finite (resp. infinite) sequences $a_1, a_2, \ldots, a_k$ (resp. $a_1, a_2, \ldots, a_k, \ldots$) of elements in Σ. The empty sequence is written ε and we denote by Σ^+ the set $\Sigma^* \setminus \{\varepsilon\}$.

2.1 Labeled Transition Systems

Definition 2.1. *A* labeled transition system *(LTS) is a* 4*-tuple* $\mathcal{S} = (S, s^0, \Sigma, \rightarrow)$ *where* S *is a set of* states*,* $s^0 \in S$ *is an* initial state*,* Σ *is a set of* labels *and* $\rightarrow \subseteq S \times \Sigma \times S$ *is a* transition relation*.*

A transition $(s, a, s') \in \rightarrow$ is also written $s \xrightarrow{a} s'$. We also write $s \xrightarrow{a}$ whenever there exists s' such that $s \xrightarrow{a} s'$. A *finite path* (resp. *infinite path*) in $\mathcal{S}$ is any finite (resp. infinite) sequence $\pi = s_1 \xrightarrow{a_1} s'_1, s_2 \xrightarrow{a_2} s'_2, \ldots, s_k \xrightarrow{a_k} s'_k, \ldots$ of transitions such that $s'_{i-1} = s_i$ for every index $i > 1$ in the sequence. We shortly write $\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \cdots s_k \xrightarrow{a_k} s_{k+1} \cdots$ and we say that π *starts in* s_1. A finite (resp. infinite) *execution* of $\mathcal{S}$ is any finite (resp. infinite) path starting in the initial state s^0 of $\mathcal{S}$. Slightly abusing notations, we will also write $s \xrightarrow{\varepsilon} s$ for every state s. The *reachability set* $\mathsf{post}^*_{\mathcal{S}}$ of $\mathcal{S}$ is the set of states that are visited by some execution.

We now present the composition primitive that we use to build complex systems from basic components: the synchronized product of labeled transitions [Arn94]. In a synchronized product, components must behave according to so-called synchronization vectors. Consider n labeled transition systems $\mathcal{S}_1, \ldots, \mathcal{S}_n$ with $\mathcal{S}_i = (S_i, s_i^0, \Sigma_i, \rightarrow_i)$. A *synchronization vector* is any n-tuple $\boldsymbol{v}$ in $\Sigma_\otimes = (\Sigma_1 \cup \{\varepsilon\}) \times \cdots \times (\Sigma_n \cup \{\varepsilon\})$, and a *synchronization constraint* is any subset $V \subseteq \Sigma_\otimes$ of synchronization vectors. Intuitively, a label a in a synchronization vector means that the corresponding component must take a transition labeled by a, whereas an ε means that the component must not move.

Definition 2.2. *The* synchronized product *of n labeled transition systems $\mathcal{S}_1, \ldots, \mathcal{S}_n$ with respect to a synchronization constraint V is the labeled transition system $\mathcal{S}_\otimes = (S_\otimes, \boldsymbol{s^0}_\otimes, \Sigma_\otimes, \rightarrow_\otimes)$ defined by: $S_\otimes = S_1 \times \cdots \times S_n$, $\boldsymbol{s^0}_\otimes = \langle s_1^0, \ldots, s_n^0 \rangle$ and $\boldsymbol{s} \xrightarrow{\boldsymbol{v}}_\otimes \boldsymbol{s'}$ iff $\boldsymbol{v} \in V$ and $\boldsymbol{s}(i) \xrightarrow{\boldsymbol{v}(i)} \boldsymbol{s'}(i)$ for every $1 \leq i \leq n$.*

2.2 Labeled Event Structures

Definition 2.3. *A* labeled event structure *(LES) is a 5-tuple $\mathcal{E} = (E, \leq, \#, \Sigma, l)$ where E is a set of* events*, $\leq$ is a partial order on E, $\#$ is a symmetric and irreflexive relation on E, Σ is a set of* labels*, and $l : E \rightarrow \Sigma$ is a* labeling function *satisfying:*

$i)$ *the downward closure $\geq[\{e\}]$ is finite for every $e \in E$, and*
$ii)$ *$e \# e'$ and $e' \leq e''$ implies $e \# e''$ for every $e, e', e'' \in E$.*

In the previous definition, relations $\leq$ and $\#$ are respectively called *causality* and *conflict* relations. Intuitively, an event e can occur when (1) every causal event e' with $e' \leq e$ has already occurred and (2) no conflicting event e' with $e' \# e$ has already occurred. Condition $i)$ enforces that any event has finitely many causal events, and condition $ii)$ expresses a *conflict inheritance* property.

A subset of E is called *conflict-free* if it does not contain any two events that are in conflict. A *configuration* is any conflict-free and downward-closed (w.r.t. causality) subset of E. We denote by $\mathcal{C}(\mathcal{E})$ (resp. $\mathcal{C}_f(\mathcal{E})$) the set of all configurations (resp. finite configurations) of a labeled event structure $\mathcal{E}$. For any event $e \in E$, the set $\geq[\{e\}]$ is called the *local configuration of* e (it is readily seen that this set is a finite configuration). We will shortly write $[e]$ the local configuration of e when the causality preorder is clear from the context. An event $e \in E$ is *enabled* at some configuration C, written $C \vdash e$, if $e \notin C$ and $C \cup \{e\}$ is a configuration. We say that a labeled event structure is *finitely-branching* if every finite configuration has finitely many enabled events. A variant of König's lemma (see Lemma A.1 in appendix) applies to finitely-branching labeled event structures.

Definition 2.4. *A* marking *for a labeled event structure $\mathcal{E}$ is any function M from $\mathcal{C}_f(\mathcal{E})$ to some set S.*

A *marked LES* is any pair $(\mathcal{E}, M)$ consisting of a labeled event structure $\mathcal{E}$ and a marking M for $\mathcal{E}$. We denote by $\mathcal{S}_{\mathcal{E}}^{M}$ the labeled transition system *induced by* $(\mathcal{E}, M)$ and defined by $\mathcal{S}_{\mathcal{E}}^{M} = (M[\mathcal{C}_f(\mathcal{E})], M(\emptyset), \Sigma, \rightarrow)$ where $s \xrightarrow{a} s'$ iff there exists a finite configuration C and an event e enabled at C such that $s = M(C)$, $a = l(e)$ and $s' = M(C \cup \{e\})$. Given a labeled transition system $\mathcal{S}$, a *marked LES for* $\mathcal{S}$ is any marked LES $(\mathcal{E}, M)$ such that $\mathcal{S}_{\mathcal{E}}^{M}$ coincides with the restriction of $\mathcal{S}$ to $\mathsf{post}_{\mathcal{S}}^{*}$. Remark that $(\mathcal{E}, M)$ is obviously a marked LES for $\mathcal{S}_{\mathcal{E}}^{M}$.

3 Truncation for Well-Structured Transition Systems

Well-Structured Transition Systems were introduced in [Fin90, AČJT00] as an abstract generalization of Petri nets satisfying the same *monotonicity* property, and hence enjoying nice decidability properties. It turns out that many classes of infinite-state systems are well-structured [FS01].

We will see in the next section how to algorithmically construct labeled event structures. However, a labeled event structure is infinite as soon as the underlying system has an infinite execution. Thus, we need property-preserving truncation techniques in order to decide verification problems using only a finite prefix of an event structure. In this section, we show how such techniques can be obtained when the underlying system is well-structured.

For simplicity we only focus, without loss of generality, on forward analysis techniques for well-structured transition systems. We show in appendix G how known backward analysis results on well-structured transition systems can be captured by this forward analysis. Moreover, we do not discuss effectivity issues (such as whether preorders need to be decidable, whether successor states need to be computable, etc.) since they are basically the same as in [FS01].

3.1 Synchronized Product of Well-Structured Transition Systems

Recall that our main objective is to verify complex systems obtained by (potentially nested) synchronized products of basic components. Thus, we first show that well-structure is preserved under synchronized product. Our presentation of well-structured transition systems differs from (and generalizes) the standard (non-labeled) one as we need to take care of labels.

Until the end of this sub-section, we assume that each set of labels Σ is partitioned into a set Σ_τ of *local labels* (for internal transitions) and a set Σ_γ of *global labels* (for synchronizable transitions). In order to account for this separation between internal transitions and synchronizable ones, we assume (1) that every synchronization constraint V implicitly contains the set $V_\tau = \{\langle \tau, \varepsilon, \ldots, \varepsilon \rangle, \ldots, \langle \varepsilon, \ldots, \varepsilon, \tau, \varepsilon, \ldots, \varepsilon \rangle, \ldots, \langle \varepsilon, \ldots, \varepsilon, \tau \rangle \,/\, \tau \in \Sigma_\tau\}$ of synchronization vectors, and (2) that no local label $\tau \in \Sigma_\tau$ may appear in a synchronization vector of $V \setminus V_\tau$. Naturally V_τ becomes the set of local labels of any synchronized product w.r.t. V.

A *preordered LTS* is any LTS $\mathcal{S} = (S, s^0, \Sigma, \rightarrow)$ equipped with a preorder $\preceq$ on S. We say that $\preceq$ is *compatible* (resp. *transitively compatible*, *reflexively compatible*) with

$\to$ if for every transition $s \xrightarrow{a} s'$ and $t \succeq s$ there exists $t' \succeq s'$ such that $t \xrightarrow{\sigma} t'$ for some $\sigma \in \Sigma^*$ satisfying:

$$\begin{cases} \sigma \in \Sigma_\tau^* & \text{if } a \in \Sigma_\tau \\ \sigma \in \Sigma_\tau^* \, a \, \Sigma_\tau^* & \text{otherwise} \end{cases} \quad \begin{cases} \sigma \in \Sigma_\tau^+ & \text{if } a \in \Sigma_\tau \\ \sigma \in \Sigma_\tau^* \, a \, \Sigma_\tau^* & \text{otherwise} \end{cases} \quad \begin{cases} \sigma \in \{\varepsilon\} \cup \Sigma_\tau & \text{if } a \in \Sigma_\tau \\ \sigma = a & \text{otherwise} \end{cases}$$

(compatibility) (transitive compatibility) (reflexive compatibility)

Moreover we also say that $\preceq$ is *strictly compatible* with $\to$ if both $\preceq$ and $\prec$ are compatible with $\to$ (recall that $s \prec s'$ is defined by $s \preceq s' \not\preceq s$). Of course, this strictness notion may be combined with transitive and reflexive compatibilities.

Remark 3.1. The previous definitions of compatibility coincide with the definitions given in [FS01] when $\Sigma = \Sigma_\tau$ is a singleton.

Any synchronized product $\mathcal{S}_\otimes$ of n preordered LTSs $(\mathcal{S}_1, \preceq_1), \ldots, (\mathcal{S}_n, \preceq_n)$ may be equipped with the *product preorder* $\preceq_\otimes$ defined by $\boldsymbol{s} \preceq_\otimes \boldsymbol{s'}$ iff $\boldsymbol{s}(i) \preceq_i \boldsymbol{s'}(i)$ for every $1 \leq i \leq n$. The following proposition shows that all six compatibility notions defined above are preserved under synchronized product.

Proposition 3.2. *Let $Cond$ denote any compatibility condition among $\{$(non-strict), strict$\} \times \{$(standard), transitive, reflexive$\}$. Any synchronized product of preordered LTSs with compatibility $Cond$ also has compatibility $Cond$.*

Recall that a *well-preorder* on some set U is any preorder $\preceq$ on U such that any infinite sequence $x_1, \ldots, x_k, \ldots$ of elements in U contains an increasing pair $x_i \preceq x_j$ with $i < j$. A *well-preordered LTS* is any preordered LTS $(\mathcal{S}, \preceq)$ where $\preceq$ is a well-preorder on the state set S of $\mathcal{S}$. Since the product preorder of any n well-preorders is also a well-preorder (from Higman's lemma), we obtain that well-preordering is preserved under synchronized product.

Proposition 3.3. *Any synchronized product of well-preordered LTSs is a well-preordered LTS.*

A *well-structured* LTS is any well-preordered LTS with (standard) compatibility. It follows from the two previous propositions that well-structure is preserved under synchronized product.

3.2 Finite Property-Preserving Truncation of Well-Structured LES

The intuition behind well-structure is that any state may be weakly simulated by any greater state, and thus we may forget about smaller states when performing reachability analysis. The well-preordering condition between states guarantees termination of the analysis [FS01]. We show in this sub-section how to extend these ideas to the partial-order verification of well-structured labeled transition systems.

Recall that any marked LES $(\mathcal{E}, M)$ induces a labeled transition system $\mathcal{S}_\mathcal{E}^M$. We lift the well-structure notions defined in the previous sub-section from labeled transition systems to labeled event structures. A *preordered marked LES* (resp. *well-preordered marked LES*) is any marked LES $(\mathcal{E}, M)$ equipped with a preorder (resp. well-preorder)

$\preceq$ on $M[\mathcal{C}_f(\mathcal{E})]$. Given any preordered marked LES $(\mathcal{E}, M, \preceq)$, we say that $(\mathcal{E}, M, \preceq)$ has compatibility $Cond \in \{\text{(non-strict), strict}\} \times \{\text{(standard), transitive, reflexive}\}$ whenever $\mathcal{S}_{\mathcal{E}}^{M}$ has compatibility $Cond$.

Consider any preordered marked LES $(\mathcal{E}, M, \preceq)$ where $\mathcal{E} = (E, \leq, \#, \Sigma, l)$. A *cutoff event* is any $e_{cut} \in E$ such that $M([e_{cut}]) \succeq M([e])$ for some event e with $e < e_{cut}$. The *truncation* $\mathcal{T}(\mathcal{E}, M, \preceq)$ of $(\mathcal{E}, M, \preceq)$ is the set of events having no strictly causal cutoff event, formally $\mathcal{T}(\mathcal{E}, M, \preceq) = E \setminus \{e \in E \,/\, \exists\, e_{cut} \in E_{cut}, e_{cut} < e\}$ where E_{cut} denotes the set of cutoff events in $\mathcal{E}$. Observe that $\mathcal{T}(\mathcal{E}, M, \preceq)$ is downward-closed, and that any minimal cutoff event (i.e. any event in $\mathsf{Min}_{\leq}(E_{cut})$) is a maximal event of $\mathcal{T}(\mathcal{E}, M, \preceq)$ but the converse does not hold in general. In order to preserve termination and boundedness properties, this truncation criterion "respects" causality, and this leads to larger truncations than in [McM95] where the truncation only preserves reachability properties.

We will show in the rest of this sub-section how to use the truncation to decide several verification problems. Unfortunately the truncation may be infinite in general, as it may be "too deep" and / or "too wide". A well-preordering condition avoids the first possibility, and a branching finiteness assumption eliminates the second.

Proposition 3.4. *The truncation of any well-preordered finitely-branching marked LES is finite.*

Given any labeled transition system $\mathcal{S}$, we say that $\mathcal{S}$ *terminates* (resp. is *bounded*) if $\mathcal{S}$ has no infinite execution (resp. has a finite reachability set $\mathsf{post}_{\mathcal{S}}^*$). The two following propositions show that, assuming an adequate compatibility condition, the truncation defined above contains enough information to decide termination and boundedness. Remark that in these two propositions, the finiteness requirement on the truncation can be dropped when the marked LES is finitely-branching and well-preordered.

Proposition 3.5. *For any preordered finitely-branching marked LES $(\mathcal{E}, M, \preceq)$ with transitive compatibility, $\mathcal{S}_{\mathcal{E}}^{M}$ terminates iff $\mathcal{T}(\mathcal{E}, M, \preceq)$ is finite and contains no cutoff event.*

In order to decide boundedness, we will need "strict" cutoff events, and we will also require a partial-order $\preceq$. Formally, a *strict cutoff event* is any $e_{cut} \in E$ such that $M([e_{cut}]) \succ M([e])$ for some event e with $e < e_{cut}$. Observe that any strict cutoff event is also a cutoff event. A *partially-ordered marked LES* is any preordered marked LES $(\mathcal{E}, M, \preceq)$ where $\preceq$ is a partial order on $M[\mathcal{C}_f(\mathcal{E})]$. Notice that the following proposition does not hold for general preordered marked LES.

Proposition 3.6. *For any partially-ordered marked LES $(\mathcal{E}, M, \preceq)$ with strict compatibility, $\mathcal{S}_{\mathcal{E}}^{M}$ is bounded iff $M[\{C \in \mathcal{C}_f(\mathcal{E}) \,/\, C \subseteq \mathcal{T}(\mathcal{E}, M, \preceq)\}]$ is finite and $\mathcal{T}(\mathcal{E}, M, \preceq)$ contains no strict cutoff event.*

We now turn our attention to the quasi-liveness problem which, assuming an adequate compatibility condition, reduces to the computation of the upward closure of $\mathsf{post}_{\mathcal{S}_{\mathcal{E}}^{M}}^*$. For any labeled transition system $\mathcal{S} = (S, s^0, \Sigma, \rightarrow)$, we say that a given label $a \in \Sigma$ is *quasi-live* if there is an execution in $\mathcal{S}$ containing a transition labeled with a.

The truncation that we have used so far would be sufficient to decide quasi-liveness, but in order to improve efficiency, we consider a refined notion of cutoff events which leads to smaller truncations (that still contain enough information to decide quasi-liveness). This refined notion is based on the size of configurations as in [McM95]. Formally, given any preordered marked LES $(\mathcal{E}, M, \preceq)$ where $\mathcal{E} = (E, \leq, \#, \Sigma, l)$, we denote by $\trianglelefteq$ the preorder on $\mathcal{C}_f(\mathcal{E})$ defined by $C \trianglelefteq C'$ iff $\mathsf{Card}(C) \leq \mathsf{Card}(C')$. Note that $C \lhd C'$ means $\mathsf{Card}(C) < \mathsf{Card}(C')$. A $\lhd$-*cutoff event* is any $e_{cut} \in E$ such that $M([e_{cut}]) \succeq M([e])$ for some event e with $[e] \lhd [e_{cut}]$. The $\lhd$-*truncation* $\mathcal{T}_\lhd(\mathcal{E}, M, \preceq)$ of $(\mathcal{E}, M, \preceq)$ is the set of events having no strictly causal $\lhd$-cutoff event, formally $\mathcal{T}_\lhd(\mathcal{E}, M, \preceq) = E \setminus \{e \in E \,/\, \exists\, e_{cut} \in E^{\lhd}_{cut}, e_{cut} < e\}$ where $E^{\lhd}_{cut}$ denotes the set of $\lhd$-cutoff events in $\mathcal{E}$.

For clarity, any (standard) cutoff event will now be called a $\subset$-cutoff event, and the (standard) truncation will now be called the $\subset$-truncation and be denoted by $\mathcal{T}_\subset(\mathcal{E}, M, \preceq$). It is readily seen that $\mathcal{T}_\lhd(\mathcal{E}, M, \preceq) \subseteq \mathcal{T}_\subset(\mathcal{E}, M, \preceq)$. Hence $\lhd$-truncations are also finite for well-preordered finitely-branching marked LESs. Notice that the following proposition requires reflexive compatibility of the inverse preorder $\succeq$ of $\preceq$ (this requirement was called "downward compatibility" in [FS01]).

Proposition 3.7. *For any preordered marked LES $(\mathcal{E}, M, \succeq)$ with reflexive compatibility, the two following assertions hold:*

$i)$ *the sets $M[\{C \in \mathcal{C}_f(\mathcal{E}) \,/\, C \subseteq \mathcal{T}_\lhd(\mathcal{E}, M, \preceq)\}]$ and $\mathsf{post}^*_{\mathcal{S}^M_\mathcal{E}}$ have the same upward closure w.r.t. $\preceq$.*

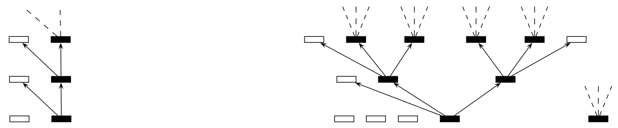
$ii)$ *for any global label a, a is quasi-live in $\mathcal{S}^M_\mathcal{E}$ iff a labels an event in $\mathcal{T}_\lhd(\mathcal{E}, M, \preceq)$.*

Remark that the previous proposition also holds for the standard truncation (i.e. we may replace $\mathcal{T}_\lhd$ by $\mathcal{T}_\subset$ in the proposition). We may even further refine the truncation by considering a preorder on $\mathcal{C}_f(\mathcal{E})$ that refines $\trianglelefteq$ (i.e. a preorder that is contained in $\trianglelefteq$). However Proposition 3.7 may not hold for this refined preorder unless we assume stronger requirements on the preordered marked LES $(\mathcal{E}, M, \succeq)$. In particular, if every label is global then Proposition 3.7 still holds for the lexicographic preorder between configurations defined in [ERV02].

4 Compositional Unfoldings of Concurrent Systems

We now give algorithms for unfolding given systems into labeled event structures. Figure 1(a) depicts an LES $\mathcal{E}_a$ modeling a positive counter initialized to 1. Black (resp. white) events represent increasing $(+)$ events (resp. decreasing $(-)$ events) and arrows represent the causality relation. Since this counter is initialized to 1, both $-$ and $+$ are initially enabled, however one needs to first unfold a $+$ event before unfolding a second $-$, and so on. Thus, unfolding $\mathcal{E}_a$ is achieved by first building the lowest two events (initialization phase), and then extending every $+$ event with new $-$ and $+$ events (extension phase).

All our unfolding algorithms rely on this principle. The following Unfold builds on-the-fly LES for given systems:



(a) (b)

Fig. 1. LES for counters with: (a) $v_0 = 1$ and $k = 1$, (b) $v_0 = 3$ and $k = 2$.

```
Unfold()
  PE:=Init()
  for (P, A) ∈ PE do
    NewPE:=Extend(P, A)
    PE:= (PE \ {(P, A)}) ∪ NewPE
  end
```

Pairs (P, A) correspond to new extensions: P is the preset of the new event (e.g. the lowest black event in Figure 1(a)) and A is the set of actions to extend with (e.g. $\{+, -\}$). Extending creates new events using the NewEvent function that also updates causality and conflict relations. Then Unfold computes new pending extensions. Notice that this algorithm terminates if Extend eventually always returns an empty set, which is the case for well-structured LESs if we do not extend cut-off events as defined in section 3.2.

In the sequel, we detail Init and Extend functions for three types of systems. We first consider *counters* for which we give dedicated functions. Ad hoc algorithms are always more efficient and can be defined for other datatypes for instance FIFO queues [LI05]. However, it is not always possible nor wanted to have specific algorithms, hence in section 4.2 we define functions that compute a concurrent marked LES for any given LTS. Finally, in section 4.3, we consider the unfolding of synchronized products of systems.

4.1 Unfolding Counters

A *counter* is a datatype with values ranging over the set of natural numbers $\mathbb{N}$, equipped with two operations: $+$ and $-$ that respectively increase and decrease its value, and initial value $v_0 \in \mathbb{N}$. It may be viewed as an LTS $\mathcal{S}_c = (\mathbb{N}, v_0, \{+, -\}, \rightarrow)$ where $n \xrightarrow{+} n + 1$ for any $n \in \mathbb{N}$ and $n \xrightarrow{-} n - 1$ for all $n > 0$. Places of Petri nets are examples of such counters.

We aim at defining Init and Extend functions that build an LES for a counter. Figure 1 depicts two different LESs $\mathcal{E}_a$ and $\mathcal{E}_b$ modeling a counter. The labeling l_c associates $+$ (resp. $-$) to every black (resp. white) event and the natural marking M_c associates to every $C \in \mathcal{C}_f(\mathcal{E})$ the value $v_0 + \mathsf{Card}(\{e \in C \,/\, l_c(e) = +\}) - \mathsf{Card}(\{e \in C \,/\, l_c(e) = -\})$. Both $(\mathcal{E}_a, M_c)$ and $(\mathcal{E}_b, M_c)$ are marked LESs for $\mathcal{S}_c$.

In these LESs, causality between $-$ and $+$ events correspond to intuitive constraints: a counter must be increased before being decreased. However, if $v_0 > 0$, it may be decreased v_0 times without any increasing. Also, $+$ events are concurrent since there is no constraint for increasing. Hence, labeled event structures $\mathcal{E}_a$ and $\mathcal{E}_b$ differ in the degree of concurrency between $+$ events. Choosing the degree $k \geq 1$ of concurrency is a matter of modeling leading to more or less concurrent truncations depending on the system that is analysed, in particular for synchronized products of LESs (see section 4.3).

Init creates v_0 $(\emptyset, \{-\})$ and k $(\emptyset, \{+\})$ pending extensions. Then, Extend simply follows the the principle depicted in Figures 1(a) and 1(b).

```
Extend (P, A)
  if (− ∈ A)  e⁻:=NewEvent(−, P)
  if (+ ∈ A)  for i ∈ [1; k] do e_i⁺:=NewEvent(+, P)
  return {({e_i⁺}, {+, −}) / i ∈ [1; k]}
```

Using our algorithm, one obtains the $(v_0 = 1, k = 1)$ counter LES in Figure 1(a), which corresponds to McMillan's unfolding of a counter [McM95]. However, Figure 1(b) shows that our approach yields the ability to choose more or less concurrent models using parameter k.

4.2 Unfolding Labeled Transition Systems

Defining the semantics of given systems as LESs or designing dedicated unfolding algorithms for those systems is often very hard. However, most systems can easily be described as LTSs. Hence, being able to compute a marked LES for any LTS is a solution to benefit from intrinsic concurrency in those systems.

A trivial LES for any LTS is its reachability tree, however every event in a reachability tree is either in causality or in conflict with any other event. We introduce an algorithm that computes a *concurrent* marked LES for any given LTS. Figure 2(b) depicts a prefix of the LES $\mathcal{E}_f$ computed by our algorithm for a FIFO queue LTS $\mathcal{S}_f$ over messages $\{a, b\}$. Concurrency essentially corresponds to independence diamonds in $\mathcal{S}_f$: whenever two or more actions are commutative. Moreover, our algorithm infers *local* concurrency: the same actions can be concurrent in some state of $\mathcal{S}_f$ and conflicting in some other state.

Init defines initially pending extension $(\emptyset, \Sigma)$ and marking $M(\emptyset) = s^0$ for the given LTS $(S, s^0, \Sigma, \rightarrow)$. Assume that e_0 in Figure 2(b) has not been extended so far: $P = \{e_0\}$ and $A = \{?a, ?b, !a, !b\}$. Extending P results in creating new events $\{e_2, e_3, e_4\}$ in causality with e_0 ($?b$ is not enabled in $M(\{e_0\})$). Now, extending $P = \{e_0, e_2\}$ with label $!a$ does not create any event since adding e_3 to P yields the expected extension. Hence, our Extend function first looks for concurrent events that can extend P, and

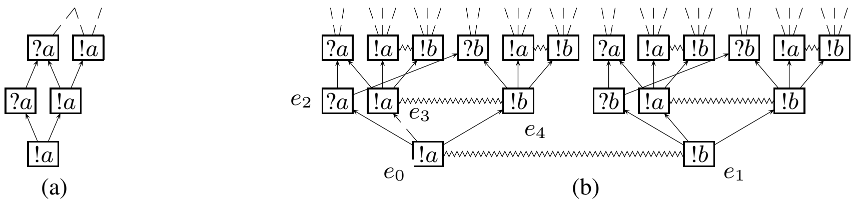


Fig. 2. LES for $\emptyset$-initialized FIFO channels with messages: (a) $\{a\}$ and (b) $\{a, b\}$.

then creates new events only for the labels in $A \setminus l(X)$ that were not matched by this first step.

```
Extend (P, A)
  X:=∅
  for e ∈ E s.t. l(e) ∈ A and P ⊢ e do X:=X ∪ e
  for a ∈ A \ l(X) s.t. M(P) -a-> do e:=NewEvent(a, P); X:=X ∪ {e}
  for e ∈ X do C(E):=C(E) ∪ {P ∪ {e}}; M(C ∪ {e}):=→[M(P), l(e)]
  return {(P ∪ {e}, Σ) / e ∈ X}
```

Notice that in this algorithm, P is always a configuration: Extend explores the configuration space of the LES.

However, Extend is not correct so far as it does not add any conflict whereas Figure 2(b) clearly shows the need for it. Missing conflicts are detected as follows. Assume that $\mathcal{E}_f$ in Figure 2(b) only contains e_0 and e_1 without conflict so far. Extending $(\{e_0\}, !b)$ leads to configuration $\{e_0, e_1\}$ with $M(\{e_0, e_1\}) = \rightarrow[M(\{e_0\}), !b] = ab$. Next, extending $(\{e_1\}, !a)$, leads to associating e_0 to $\{e_1\}$ which results to be impossible since $\rightarrow[M(\{e_1\}), !a] = ba \neq M(\{e_0, e_1\})$. Hence, conflict must be added between e_0 and e_1 using the CheckConflict function below when Extend detects the problem.

```
CheckConflict (P, PE)
  for e ∈ E s.t. P ⊢ e and (M(P) -l(e)-/-> or M(P ∪ {e}) ≠ →[M(P), l(e)]) do
    e':=choose in Max≤(P)
    E:=E \ {e'' ∈ E / e ≤ e'' and e' ≤ e''}
    C(E):=C(E) \ {C ∈ C(E) / {e, e'} ∈ C}
    PE:=(PE ∩ C(E)) ∪ {(P', Σ) / P' ∈ C(E), (e ∈ P', P' ⊢ e') or (e' ∈ P', P' ⊢ e)}
    #:=# ∪ {⟨e, e'⟩, ⟨e', e⟩}
  end
  return Sort(PE)
```

CheckConflict updates PE since whenever one needs to add conflict between 2 events e_0 and e_1, every configuration in $\mathcal{C}(\mathcal{E})$ that contains both events must be discarded and every configuration that contains e_0 (resp. e_1) has potentially mistaken extensions. Notice that pending extensions (P, A) in PE are eventually sorted w.r.t increasing size of P. This is due to a natural hypothesis made by Extend: if P is to be extended, then all the extensions of any $P' \subset P$ are up-to-date.

Figure 2 depicts the marked LES obtained for LTS modeling FIFO queues in the standard way (one state per queue content, and transitions w.r.t. FIFO policy) by applying our algorithm. They exactly correspond to the LES computed by the method in [LI05].

4.3 Unfolding Synchronized Products of Components

Sections 4.1 and 4.2 present unfolding algorithms for single components. We now introduce an algorithm for unfolding complex systems built from synchronized components.

Consider Petri net N in Figure 3(a). In our framework, each place p_i is modeled by a counter LES and each transition t_j by a synchronization vector between actions of

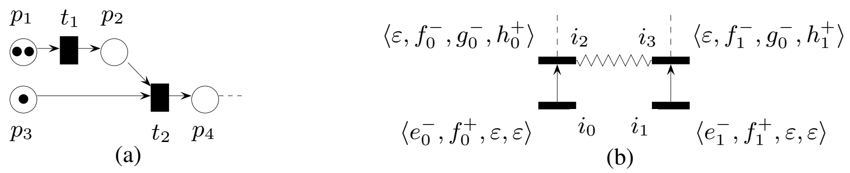


Fig. 3. A Petri net N (a) and a marked LES for N (b).

these counters. Since tokens in Petri nets are concurrent processes, we choose a ($v_0 = 2, k = 2$) counter LES $\mathcal{E}_1$ for p_1 since it initially contains 2 tokens. Similarly we choose a $(1, 1)$ counter LES $\mathcal{E}_3$ for p_3. Place p_2 is initially empty and can simultaneously contain 2 tokens, thus we model it by a $(0, 2)$ counter LES $\mathcal{E}_2$. Finally, we choose a $(0, 1)$ counter LES $\mathcal{E}_4$ for p_4. In the case of unbounded places, one can choose k as the number of entering edges.

Let e_i^a (resp. f_i^a, g_i^a and h_i^a) denote the ith event labeled by $a \in \{+, -\}$ in $\mathcal{E}_1$ (resp. $\mathcal{E}_2$, $\mathcal{E}_3$ and $\mathcal{E}_4$) w.r.t. causality. The semantics of N is modeled in the synchronized product of $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$ and $\mathcal{E}_4$ by the synchronization vectors $\langle -, +, \varepsilon, \varepsilon\rangle$ for t_1 and $\langle \varepsilon, -, -, +\rangle$ for t_2.

Figure 3(b) depicts LES $\mathcal{E}_N$ obtained for N using our unfolding algorithm. To each event in $\mathcal{E}_N$ is associated a tuple of components' events by mapping $\lambda : (E_1 \cup \{\varepsilon\}) \times \cdots \times (E_n \cup \{\varepsilon\}) \rightarrow E$, for instance $\lambda(i_0) = \langle e_0^-, f_0^+, \varepsilon, \varepsilon\rangle$. Conflict and causality relations in $\mathcal{E}_N$ are defined from components' ones. Basically, conflict appears when a components' event is used by two or more global events, e.g. g_0^- in i_2 and i_3, and causality inherits from components, e.g. $f_0^+ \rightarrow f_0^-$ entails $i_0 \rightarrow i_2$. Formally, let $\langle e_1, \ldots, e_n\rangle \# \langle e'_1, \ldots, e'_n\rangle$ iff there exists i s.t. $e_i = e'_i$ or $e_i \#_i e'_i$. The global causality and conflict relations are respectively the smallest partial order $\leq$ and the smallest symmetric and irreflexive relation $\#$ satisfying for every global events e, e', e'':

- if $e \# e'$ and $e' \leq e''$ then $e \# e''$, and
- if $\lambda(e) \# \lambda(e')$ then $e \# e'$, and
- if there exists i s.t. $(\lambda(e))_i \leq_i (\lambda(e'))_i$ and we do not have $e \# e'$ then $e \leq e'$.

As Figure 3(b) shows, unfolding a synchronized product of LESs consists in associating components' events into global events w.r.t. synchronization vectors, conflict and causality relations. Since components' LES maybe infinite we use an on-the-fly algorithm that proceeds as follows. Init initializes every component (in particular PE_i) and extends *all* their initially pending extensions $(\emptyset, A_i)$. This is necessary due to synchronization. Next, extending (P, A) in the global LES consists, for every synchronization vector $v \in A$, in finding all tuples $\langle e_1, \ldots, e_n\rangle$ of components' events which are instances of v that extend P. A new global event e is created for each such instance $\langle e_1, \ldots, e_n\rangle$ and each conflict-free preset ps of global events that match the presets of every e_i. Finally, every component such that $e_i \neq \varepsilon$ is extended since the successors of e_i may be needed to extend further.

```
Extend (P, A)
  NewE := ∅
  if (P = ∅)
```

```
    E⊗ :={⟨e1,...,en⟩ / ⟨l(e1),...,l(en)⟩ ∈ A  and  ei ∈ Min≤i(Ei) ∪ {ε}}
  else
    E⊗ :={⟨e1,...,en⟩ / ⟨l(e1),...,l(en)⟩ ∈ A and ∃e′ ∈ Max≤(P), ∃i, ei ∈ (λ(e′))i•}
  for ⟨e1,...,en⟩ ∈ E⊗ do
    for ps ∈ {E′ ∈ 2^E / ∀e,e′ ∈ E′, e #e′  and ∀i, (λ(E′))i = •ei} do
      e := NewEvent(⟨l(e1),...,l(en)⟩, ps)
      λ(e) := ⟨e1,...,en⟩
      for i ∈ [1;n] s.t. l(ei) ≠ ε and ei• = ∅ do PEi :=Extendi(PEi, Σi)
      NewE := NewE ∪ {e}
    end
  end
  return {(≥[e], Σ⊗) / e ∈ NewE}
```

In this algorithm, we denote by $\bullet e = \mathsf{Max}_{\leq}((\geq[e]) \setminus \{e\})$ the preset of e w.r.t. causality, and by $e\bullet = \mathsf{Min}_{\leq}((\leq[e]) \setminus \{e\})$ the postset of e. Σ_i denotes the set of actions of component i. Notice that $\mathsf{Extend}_i(PE_i, \Sigma_i)$ is a slight abuse of notations as PE_i is a *set* of pending extensions.

Extend first checks that components' events have not been extended yet before doing so ($e_i\bullet = \emptyset$) since an event may be associated to many global events. The labeling of global events and configurations are defined component-wise, and global conflict and causality relations are computed as defined previously.

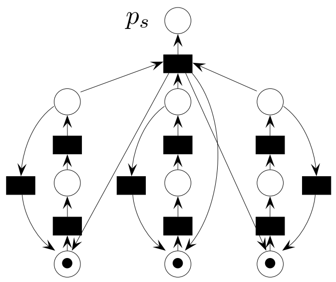
Using our algorithm, one can compute a marked unfolding $\mathcal{E}_\otimes$ of a synchronized product of components as depicted in Figure 3. Furthermore, $\mathcal{E}_\otimes$ can itself be used as a component, giving raise to hierarchical unfolding of systems and components.

5 Experimental Results

We have implemented the algorithms and truncation techniques presented in this paper in a tool called ESU. This tool is implemented in Objective Caml, and permits the verification of termination, boundedness and quasi-liveness for synchronized products of well-structured components. Components may be counters, queues or finite-state (control) automata. For the particular case of bounded systems such as bounded Petri nets, ESU is also able to compute reachability set.

To our knowledge, ESU is the first tool able to analyse infinite-state systems using forward unfolding techniques. Hence, in order to evaluate the benefits of our approach we have compared ESU with two tools for Petri nets: the PEP environment which provides an unfolding tool for bounded Petri nets [Pep], and the tool TINA which analyzes arbitrary Petri nets using structural analysis techniques and forward Karp-Miller reachability analysis [Tin]. Petri nets are modeled in ESU by synchronized counter components. Experiments were conducted on an Intel XEON 2.2 GHz station with 6 GB of RAM. In the following tables, E (resp. E_{cf}, N, S) denotes the number of events in the truncation (resp. of cutoff events, of nodes in TINA's tree, of markings in TINA's tree), and a '–' means that the analysis exhausted memory or did not finish within 10 minutes.

The Petri net depicted below represents a concurrent Producer/Consumer Petri Net with n independent production lines and m machines on each line. The products from these n lines are combined into another product that is then stored in place p_s. PEP's unfolder cannot analyze this Petri net as it is unbounded. ESU performs very well on this example, but this is not very surprising as this Petri net is extremely concurrent. Observe that the number of events in the truncation is approximately the number of transitions in the Petri net.



	TINA		ESU		
$m \times n$	N	T(s)	E	E_{cf}	T(s)
3×3	49	0.01	10	4	0.01
5×5	4636	0.04	25	5	0.01
7×7	1094241	24.41	50	8	0.01
7×10	–	–	71	8	0.03
10×10	–	–	96	6	0.04
20×25	–	–	491	11	1.4

We also experimented on a more challenging and well-known example: the swimming pool. The swimming pool has much less explicit concurrency as most transitions share places. We used TINA's bounded swimming pool Petri net which is a variant of the classical one with an additional place that limits the number of clients [Tin]. In the following table, the size denotes the number of resources in the swimming pool.

	PEP			TINA			ESU		
Size	E	E_{cf}	T(s)	N	S	T(s)	E	E_{cf}	T(s)
3	37593	18009	159.59	126	56	0.00	18	3	0.01
10	–	–	–	12012	3003	0.05	60	10	0.20
20	–	–	–	255024	53130	3.35	120	20	3.02
30	–	–	–	1669536	324632	44.74	180	30	20.64
40	–	–	–	6516048	1221759	297.19	240	40	64.04

Future work will focus on improving and extending our method to other frameworks for the analysis of infinite state systems. In particular we plan to focus on abstraction algorithms in order to build more compact and concurrent event structures that would abstract away causality and conflict information that is irrelevant w.r.t. to a desired property. We also plan to consider acceleration techniques as a tool for truncating unfoldings, hence enforcing the termination of our algorithms while preserving reachability properties.

Acknowledgements. The authors wish to thank Igor Walukiewicz for insightful comments and suggestions on a preliminary version of this paper.

References

[AČJT00] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y. K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1–2):109–127, 2000.

[AIN00] P. A. Abdulla, S. P. Iyer, and A. Nylén. Unfoldings of unbounded petri nets. In *Proc. of 12th Int. Conf. on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 495–507. Springer, 2000.

[AJKP98] P. A. Abdulla, B. Jonsson, M. Kindahl, and D. Peled. A general approach to partial order reductions in symbolic verification (extended abstract). In *Proc. of 10th Int. Conf. on Computer Aided Verification (CAV '98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 1998.

[Arn94] A. Arnold. *Finite Transition Systems. Semantics of Communicating Systems*. Prentice Hall Int., 1994.

[CGP01] J-M. Couvreur, S. Grivet, and D. Poitrenaud. Unfolding of products of symmetrical petri nets. In *Proc. 22nd Int. Conf. on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 121–143. Springer, 2001.

[DJN04] J. Desel, G. Juhás, and C. Neumair. Finite unfoldings of unbounded petri nets. In *Proc. 25th Int. Conf. on Applications and Theory of Petri Nets (ICATPN'04)*, volume 3099 of *Lecture Notes in Computer Science*, pages 157–176. Springer, 2004.

[ER99] J. Esparza and S. Römer. An unfolding algorithm for synchronous products of transition systems. In *10th Int. Conf. on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 1999.

[ERV02] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.

[Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.

[FS01] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[God96] P. Godefroid. *Partial-order methods for the verification of concurrent systems: An approach to the state-explosion problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, New York, NY, USA, 1996.

[KK03] V. Khomenko and M. Koutny. Branching processes of high-level petri nets. In *Proc. 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 458–472. Springer, 2003.

[LI05] Y. Lei and S. P. Iyer. An approach to unfolding asynchronous communication protocols. In *Proc. 13th Int. Symp. on Formal Methods (FM'05)*, volume 3582 of *Lecture Notes in Computer Science*, pages 334–349. Springer, 2005.

[Maz86] A. W. Mazurkiewicz. Trace theory. In *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 279–324. Springer, 1986.

[McM95] K. L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–45, 1995.

[NPW81] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.

[NRT95] M. Nielsen, G. Rozenberg, and P. S. Thiagarajan. Transition systems, event structures and unfoldings. *Information and Computation*, 118(2):191–207, 1995.

[Pel93] D. Peled. All from one, one for all: on model checking using representatives. In *Proc. of the 5th Int. Conf. on Computer Aided Verification (CAV'93)*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993.

[Pep] PEP tool. *Homepage:* `http://peptool.sourceforge.net/`.

[Tin] TINA tool. *Homepage:* `http://www.laas.fr/tina/`.

[Val91] A. Valmari. Stubborn sets for reduced state space generation. In *Proc. of 10th Int. Conf. on Applications and Theory of Petri Nets (ICATPN'90)*, number 483 in Lecture Notes in Computer Science. Springer, 1991.

A Preliminary notations and lemmas

Consider any relation $R \subseteq U \times U$. We shortly write $x\,R\,x'$ whenever $(x, x') \in R$. The relation $U \times U \setminus R$ is denoted by $\not R$. We say that R is *reflexive* (resp. *irreflexive*, *antisymmetric*, *transitive*) if $x\,R\,x$ (resp. $x \not R\, x$, $x\,R\,y$ and $y\,R\,x$ implies $x = y$, $x\,R\,y$ and $y\,R\,z$ implies $x\,R\,z$) for every $x, y, z \in U$. We denote by R^* the *reflexive transitive closure* of R (the smallest reflexive and transitive relation containing R). In the paper, binary relations are also shortly called *relations*.

We write $\mathsf{len}(\sigma)$ for the *length* of any finite sequence $\sigma \in \Sigma^*$. Consider any labeled event structure $\mathcal{E} = (E, \leq, \#, \Sigma, l)$. By *linearization* of any configuration $C \in \mathcal{C}(\mathcal{E})$ we mean any finite or infinite sequence $e_1, e_2, \ldots, e_k, \ldots$ of distinct events in E such that $C = \{e_1, e_2, \ldots, e_k, \ldots\}$ and $e_i \leq e_j$ implies $i \leq j$ for all indices i, j in the sequence. The following lemma states an “event structure variant” of König’s lemma (which applies to trees).

Lemma A.1. *Every downward-closed infinite subset of events in a finitely-branching labeled event structure contains an infinite configuration.*

Proof. Consider a finitely-branching labeled event structure $\mathcal{E}$ with a set E of events and assume that $X \subseteq E$ is infinite and downward-closed. Let $L \subseteq E^*$ denote the set of all linearizations of configurations included in X. Observe that L is prefix-closed (w.r.t. to the usual prefix partial order between sequences) and hence we may organize L into a (directed rooted) tree T. Moreover L is infinite since distinct local configurations have disjoint linearizations, and hence T is an infinite tree. Branching finiteness of $\mathcal{E}$ implies that T is finitely-branching, thus we obtain from König’s lemma that T contains an infinite branch. Hence there exists an infinite linearization $e_1 \cdots e_k \cdots$ in L and the set $\{e_1, \ldots, e_k, \ldots\}$ is an infinite configuration contained in X. $\square$

The two following lemmas directly follow from the definition of $\mathcal{S}^M_{\mathcal{E}}$ and express strong relationships (that will be useful in the following proofs) between a marked LES and its induced labeled transition system.

Lemma A.2. *For any two finite configurations $C \subseteq C'$ of a marked LES $(\mathcal{E}, M)$, there is a path $M(C) \xrightarrow{l(e_1)\cdots l(e_k)} M(C')$ in $\mathcal{S}^M_{\mathcal{E}}$ for every linearization $e_1, \ldots, e_k$ of $C' \setminus C$.*

Lemma A.3. *For any finite configuration C in a marked LES $(\mathcal{E}, M)$ and for any path $M(C) \xrightarrow{\sigma} s'$ in $\mathcal{S}^M_{\mathcal{E}}$, there exist a finite configuration $C' \supseteq C$ and a linearization $e_1, \ldots, e_k$ of $C' \setminus C$ such that $s' = M(C')$ and $\sigma = l(e_1) \cdots l(e_k)$.*

B Proof of Proposition 3.2

Proposition 3.2. *Let $Cond$ denote any compatibility condition among $\{$(non-strict), strict$\} \times \{$(standard), transitive, reflexive$\}$. Any synchronized product of preordered LTSs with compatibility $Cond$ also has compatibility $Cond$.*

Proof. Consider n preordered LTSs $(\mathcal{S}_1, \preceq_1), \ldots, (\mathcal{S}_n, \preceq_n)$ with $\mathcal{S}_i = (S_i, s_i^0, \Sigma_i, \rightarrow_i)$, and assume that each $\mathcal{S}_i$ has compatibility $Cond$. We show that $(\mathcal{S}_\otimes, \preceq_\otimes)$ has compatibility $Cond$, where $\mathcal{S}_\otimes$ is the synchronized product of $\mathcal{S}_1, \ldots, \mathcal{S}_n$ with respect to a given synchronization constraint V. Assume that $\boldsymbol{s} \xrightarrow{\boldsymbol{v}} \boldsymbol{s'}$ and $\boldsymbol{t} \succeq_\otimes \boldsymbol{s}$. There are two cases, depending on whether $\boldsymbol{v}$ contains a local label:

1. $\boldsymbol{v}(j) \in \Sigma_{j_\tau}$ for some $1 \leq j \leq n$ and $\boldsymbol{v}(i) = \varepsilon$ for all $i \neq j$. It follows from compatibility of $\mathcal{S}_j$ that there exists a path $\pi_j = \boldsymbol{t}(j) \xrightarrow{a_1}_j u_2 \xrightarrow{a_2}_j u_3 \cdots \xrightarrow{a_k}_j u_{k+1}$ with $u_{k+1} \succeq_j \boldsymbol{s'}(j)$ and $a_i \in \Sigma_{j_\tau}$. This path may be extended to the synchronized product as $\pi = \boldsymbol{t} \xrightarrow{\boldsymbol{v_1}}_\otimes \boldsymbol{t_2} \xrightarrow{\boldsymbol{v_2}}_\otimes \boldsymbol{t_3} \cdots \xrightarrow{\boldsymbol{v_k}}_\otimes \boldsymbol{t_{k+1}}$ where $\boldsymbol{v_h} = \langle \varepsilon, \ldots, \varepsilon, a_i, \varepsilon, \ldots, \varepsilon \rangle$, $\boldsymbol{t_h}(j) = u_h$ and $\boldsymbol{t_h}(i) = \boldsymbol{t}(i)$ for all $i \neq j$. Observe that $\boldsymbol{t_{k+1}} \succeq \boldsymbol{s'}$ since $\boldsymbol{s'}(i) = \boldsymbol{s}(i) \preceq \boldsymbol{t}(i)$ for all $i \neq j$.
2. $\boldsymbol{v}(i) \in \{\varepsilon\} \cup \Sigma_{i\gamma}$ for all $1 \leq i \leq n$. We may assume without loss of generality that there exists $1 \leq m \leq n$ such that $\boldsymbol{v}(i) \in \Sigma_{i\gamma}$ for all $i \leq m$ and $\boldsymbol{v}(i) = \{\varepsilon\}$ for all $m < i$. From compatibility of $\mathcal{S}_i$ with $i \leq m$, we obtain that there exists m paths $\pi_i = \boldsymbol{t}(i) \xrightarrow{\sigma_i}_i u_i \xrightarrow{\boldsymbol{v}(i)}_i u_i' \xrightarrow{\sigma_i'}_i t_i'$ with $t_i' \succeq_i \boldsymbol{s'}(i)$ and $\sigma_i, \sigma_i' \in \Sigma_{i\tau}^*$. Remark that $\boldsymbol{u} \xrightarrow{\boldsymbol{v}}_\otimes \boldsymbol{u'}$ where $\boldsymbol{u}(i) = u_i$ and $\boldsymbol{u'}(i) = u_i'$ for all $i \leq m$, and $\boldsymbol{u}(i) = \boldsymbol{u'}(i) = \boldsymbol{t}(i)$ otherwise. As in the previous case, we may extend each sub-path $\boldsymbol{t}(i) \xrightarrow{\sigma_i}_i u_i$ and $u_i' \xrightarrow{\sigma_i'}_i t_i'$ to the synchronized product, and their concatenations yields a path $\pi = \boldsymbol{t} \xrightarrow{\sigma_\otimes}_\otimes \boldsymbol{u} \xrightarrow{\boldsymbol{v}}_\otimes \boldsymbol{u'} \xrightarrow{\sigma_\otimes'}_\otimes \boldsymbol{t'}$ where $\sigma_\otimes, \sigma_\otimes' \in V_\tau^*$, $\boldsymbol{t'}(i) = t_i'$ for all $1 \leq i \leq m$ and $\boldsymbol{t'}(i) = \boldsymbol{t}(i)$ otherwise. Observe that $\boldsymbol{t'} \succeq \boldsymbol{s'}$ since $\boldsymbol{s'}(i) = \boldsymbol{s}(i) \preceq \boldsymbol{t}(i)$ for all $i > m$.

Thus we obtain that there exists $\boldsymbol{t'} \succeq \boldsymbol{s'}$ and $\sigma \in V^*$ such that $\boldsymbol{t} \xrightarrow{\sigma}_\otimes \boldsymbol{t'}$. Moreover, a routine check shows that in both cases, the constructed path $\boldsymbol{t} \xrightarrow{\sigma}_\otimes \boldsymbol{t'}$ satisfies $Cond$'s requirements (for strict compatibility, the component path(s) should be carefully chosen so as to ensure strictness). $\square$

C Proof of Proposition 3.4

Proposition 3.4. *The truncation of any well-preordered finitely-branching marked LES is finite.*

Proof. By contradiction, consider a well-preordered finitely-branching marked LES $(\mathcal{E}, M, \preceq)$ and assume that $\mathcal{T}(\mathcal{E}, M, \preceq)$ is infinite. Since $\mathcal{T}(\mathcal{E}, M, \preceq)$ is downward-closed, we obtain from Lemma A.1 that there is an infinite configuration $C \subseteq \mathcal{T}(\mathcal{E}, M, \preceq$). Now consider a linearization $e_1 \cdots e_k \cdots$ of C. Since $\preceq$ is a well-preorder on $M[\mathcal{C}_f(\mathcal{E})]$, there exists $i < j$ such that $M([e_i]) \preceq M([e_j])$. We get that e_j is a cutoff event, and therefore $e_{j+1} \notin \mathcal{T}(\mathcal{E}, M, \preceq)$ which contradicts the fact that C is contained in $\mathcal{T}(\mathcal{E}, M, \preceq)$. $\square$

D Proof of Proposition 3.5

Proposition 3.5. *For any preordered finitely-branching marked LES* $(\mathcal{E}, M, \preceq)$ *with transitive compatibility,* $\mathcal{S}_{\mathcal{E}}^{M}$ *terminates iff* $\mathcal{T}(\mathcal{E}, M, \preceq)$ *is finite and contains no cutoff event.*

Proof. Consider any preordered finitely-branching marked LES $(\mathcal{E}, M, \preceq)$ with transitive compatibility, where $\mathcal{E} = (E, \leq, \#, \Sigma, l)$. Let us first prove implication $\Leftarrow$ and assume that $\mathcal{T}(\mathcal{E}, M, \preceq)$ is finite and contains no cutoff event. Since $\mathcal{T}(\mathcal{E}, M, \preceq)$ contains no cutoff event we get that $E = \mathcal{T}(\mathcal{E}, M, \preceq)$ and hence E is finite. We deduce from Lemma A.3 that every execution in $(\mathcal{E}, M, \preceq)$ has length at most $\mathsf{Card}(E)$ and hence $\mathcal{S}_{\mathcal{E}}^{M}$ terminates.

Let us now prove implication $\Rightarrow$ by contraposition and assume that: $\mathcal{T}(\mathcal{E}, M, \preceq)$ is infinite or contains a cutoff event. If $\mathcal{T}(\mathcal{E}, M, \preceq)$ is infinite, then according to Lemma A.1 there is an infinite configuration C in $\mathcal{E}$. Consider a linearization $e_1, \ldots, e_k, \ldots$ of C. It is readily seen that $M(\emptyset) \xrightarrow{l(e_1)} M(\{e_1\}) \xrightarrow{l(e_2)} M(\{e_1, e_2\}) \cdots M(\{e_1, \ldots e_k\}) \xrightarrow{l(e_k)} M(\{e_1, \ldots, e_{k+1}\}) \cdots$ is an infinite execution in $\mathcal{S}_{\mathcal{E}}^{M}$. If $\mathcal{T}(\mathcal{E}, M, \preceq)$ contains a cutoff event e_{cut}, then $M([e_{cut}]) \succeq M([e])$ for some event e with $e < e_{cut}$. We get that $[e] \subset [e_{cut}]$ and we deduce from Lemma A.2 that there is a path $s^0 = M(\emptyset) \xrightarrow{*} M([e]) \xrightarrow{\sigma} M([e_{cut}])$ in $\mathcal{S}_{\mathcal{E}}^{M}$ for some $\sigma \in \Sigma^+$. Transitive compatibility of $(\mathcal{S}_{\mathcal{E}}^{M}, \preceq)$ implies that for any non-empty path $s \xrightarrow{*} s'$ with $s' \succeq s$ there is a non-empty path $s' \xrightarrow{*} s''$ with $s'' \succeq s'$. By iterating this construction on the path $M([e]) \xrightarrow{\sigma} M([e_{cut}])$, we may build an infinite execution in $\mathcal{S}_{\mathcal{E}}^{M}$. □

E Proof of Proposition 3.6

Proposition 3.6. *For any partially-ordered marked LES* $(\mathcal{E}, M, \preceq)$ *with strict compatibility,* $\mathcal{S}_{\mathcal{E}}^{M}$ *is bounded iff* $M[\{C \in \mathcal{C}_f(\mathcal{E}) \,/\, C \subseteq \mathcal{T}(\mathcal{E}, M, \preceq)\}]$ *is finite and* $\mathcal{T}(\mathcal{E}, M, \preceq)$ *contains no strict cutoff event.*

Proof. Consider any partially-ordered marked LES $(\mathcal{E}, M, \preceq)$ with strict transitive compatibility, where $\mathcal{E} = (E, \leq, \#, \Sigma, l)$. Let us shortly write $\mathcal{T} = \mathcal{T}(\mathcal{E}, M, \preceq)$ and $M_{\mathcal{T}} = M[\{C \in \mathcal{C}_f(\mathcal{E}) \,/\, C \subseteq \mathcal{T}\}]$. Recall that $M[\mathcal{C}_f(\mathcal{E})]$ is the state set of $\mathcal{S}_{\mathcal{E}}^{M}$ and hence we may deduce from Lemma A.2 that $\mathsf{post}^*_{\mathcal{S}_{\mathcal{E}}^{M}} = M[\mathcal{C}_f(\mathcal{E})]$. Observe also that $M[\mathcal{C}_f(\mathcal{E})] \supseteq M_{\mathcal{T}}$.

Let us first prove implication $\Leftarrow$ and assume that $\mathcal{T}$ contains no strict cutoff event. Since $\mathsf{post}^*_{\mathcal{S}_{\mathcal{E}}^{M}} = M[\mathcal{C}_f(\mathcal{E})]$, it suffices to show that $M[\mathcal{C}_f(\mathcal{E})] \subseteq M_{\mathcal{T}}$. By contradiction, suppose that the "bad" set $\mathcal{B} = \{C \in \mathcal{C}_f(\mathcal{E}) \,/\, M(C) \notin M_{\mathcal{T}}\}$ is non-empty. Let us choose some $B \in \mathcal{B}$ with minimal size (i.e. $B \in \mathcal{B}$ and there is no $B' \in \mathcal{B}$ with $\mathsf{Card}(B') < \mathsf{Card}(B)$). We get that $B \not\subseteq \mathcal{T}$ as $M(B) \notin M_{\mathcal{T}}$, and hence there exists $e_b \in B$ and a cutoff event e_{cut} such that $e_{cut} < e_b$. Since e_{cut} is a cutoff event, we have $M([e_{cut}]) \succeq M([e])$ for some event e with $e < e_{cut}$. However e_{cut} is not a strict cutoff event, so we get that $M([e_{cut}]) = M([e])$ as $\preceq$ is a partial order. Since $[e_{cut}] \subseteq B$ we obtain from Lemma A.2 that there is a path $M([e_{cut}]) = M([e]) \xrightarrow{\sigma} M(B)$

for some sequence σ with length $\mathsf{Card}(B \setminus [e_{cut}])$. According to Lemma A.3 there exists $B' \supseteq [e]$ such that $M(B') = M(B)$ and $\mathsf{Card}(B' \setminus [e]) = \mathsf{len}(\sigma)$. Observe that $\mathsf{Card}(B') = \mathsf{Card}(B) - \mathsf{Card}([e_{cut}]) + \mathsf{Card}([e])$, and that $[e] \subset [e_{cut}]$ since $e < e_{cut}$. Therefore $B' \in \mathcal{B}$ and $\mathsf{Card}(B') < \mathsf{Card}(B)$, a contradiction as B was supposed to be an element of $\mathcal{B}$ with minimal size.

Let us now prove implication $\Rightarrow$ by contraposition and assume that: $M_{\mathfrak{T}}$ is infinite or $\mathfrak{T}(\mathcal{E}, M, \preceq)$ contains a strict cutoff event. If $M_{\mathfrak{T}}$ is infinite, then $\mathsf{post}^*_{\mathcal{S}^M_{\mathcal{E}}} = M[\mathcal{C}_f(\mathcal{E})]$ is infinite and therefore $\mathcal{S}^M_{\mathcal{E}}$ is unbounded. If $\mathfrak{T}(\mathcal{E}, M, \preceq)$ contains a strict cutoff event e_{cut}, then $M([e_{cut}]) \succ M([e])$ for some event e with $e < e_{cut}$. We get that $[e] \subset [e_{cut}]$ and we deduce from Lemma A.2 that there is a path $s^0 = M(\emptyset) \xrightarrow{*} M([e]) \xrightarrow{*} M([e_{cut}])$ in $\mathcal{S}^M_{\mathcal{E}}$. Strict compatibility of $(\mathcal{S}^M_{\mathcal{E}}, \preceq)$ implies that for any path $s \xrightarrow{*} s'$ with $s' \succ s$ there is a path $s' \xrightarrow{*} s''$ with $s'' \succ s'$. Observe also that any path $s \xrightarrow{*} s'$ with $s' \succ s$ is necessarily non-empty. By iterating this construction on the path $M([e]) \xrightarrow{*} M([e_{cut}])$, we may build an infinite execution $\pi = s^0 \xrightarrow{*} s_1 \xrightarrow{*} s_2 \cdots s_k \xrightarrow{*} s_{k+1} \cdots$ in $\mathcal{S}^M_{\mathcal{E}}$ such that $s_i \prec s_{i+1}$ for all $i \geq 1$. It follows that π visits infinitely many (distinct) states and hence $\mathcal{S}^M_{\mathcal{E}}$ is unbounded. $\square$

F Proof of Proposition 3.7

Lemma F.1. *Consider a preordered marked LES $(\mathcal{E}, M, \succeq)$ with reflexive compatibility. For any $C \in \mathcal{C}_f(\mathcal{E})$ there exists $C' \in \mathcal{C}_f(\mathcal{E})$ such that $M(C') \preceq M(C)$ and C' contains no $\lhd$-cutoff event.*

Proof. The proof is similar to the first part of the proof of Proposition 3.6, except that we manipulate upward-closed sets. Let $\mathcal{H}$ denote the set of all $C \in \mathcal{C}_f(\mathcal{E})$ such that C contains no $\lhd$-cutoff event. By contradiction, suppose that the "bad" set $\mathcal{B} = \{C \in \mathcal{C}_f(\mathcal{E}) \,/\, M(C) \notin {\preceq}[M[\mathcal{H}]]\}$ is non-empty. Let us choose some $B \in \mathcal{B}$ with minimal size (i.e. $B \in \mathcal{B}$ and there is no $B' \in \mathcal{B}$ with $\mathsf{Card}(B') < \mathsf{Card}(B)$). We get that $B \notin \mathcal{H}$ as $M(B) \notin M[\mathcal{H}]$, and hence there exists a $\lhd$-cutoff event e_{cut} in B. Since e_{cut} is a $\lhd$-cutoff event, we have $M([e_{cut}]) \succeq M([e])$ for some event e with $[e] \lhd [e_{cut}]$. Since $[e_{cut}] \subseteq B$ we obtain from Lemma A.2 that there is a path $M([e_{cut}]) \xrightarrow{\sigma} M(B)$ in $\mathcal{S}^M_{\mathcal{E}}$ for some sequence σ with length $\mathsf{Card}(B \setminus [e_{cut}])$. Reflexive compatibility of $(\mathcal{S}^M_{\mathcal{E}}, \succeq)$ implies that there is also a path $M([e]) \xrightarrow{\sigma'} s'$ for some $s' \preceq M(B)$ and σ' with length $\mathsf{len}(\sigma') \leq \mathsf{len}(\sigma)$. According to Lemma A.3 there exists $B' \supseteq [e]$ such that $M(B') = s'$ and $\mathsf{Card}(B' \setminus [e]) = \mathsf{len}(\sigma')$. It follows from $M(B') = s' \preceq M(B)$ that $B' \in \mathcal{B}$. Observe that $\mathsf{Card}(B') = \mathsf{len}(\sigma') + \mathsf{Card}([e]) \leq \mathsf{Card}(B) - \mathsf{Card}([e_{cut}]) + \mathsf{Card}([e])$, and recall that $[e] \lhd [e_{cut}]$. Therefore $B' \in \mathcal{B}$ and $\mathsf{Card}(B') < \mathsf{Card}(B)$, a contradiction as B was supposed to be an element of $\mathcal{B}$ with minimal size. $\square$

Proposition 3.7. *For any preordered marked LES $(\mathcal{E}, M, \succeq)$ with reflexive compatibility, the two following assertions hold:*

$i)$ the sets $M[\{C \in \mathcal{C}_f(\mathcal{E}) \,/\, C \subseteq \mathfrak{T}_{\lhd}(\mathcal{E}, M, \preceq)\}]$ and $\mathsf{post}^*_{\mathcal{S}^M_{\mathcal{E}}}$ have the same upward closure w.r.t. $\preceq$.

$ii)$ for any global label a, a is quasi-live in $\mathcal{S}^M_{\mathcal{E}}$ iff a labels an event in $\mathfrak{T}_{\lhd}(\mathcal{E}, M, \preceq)$.

Proof. Consider any preordered marked LES $(\mathcal{E}, M, \succeq)$ with reflexive compatibility, where $\mathcal{E} = (E, \leq, \#, \Sigma, l)$. Let us shortly write $\mathcal{T}_{\lhd} = \mathcal{T}_{\lhd}(\mathcal{E}, M, \preceq)$ and $M_{\mathcal{T}_{\lhd}} = M[\{C \in \mathcal{C}_f(\mathcal{E}) \,/\, C \subseteq \mathcal{T}_{\lhd}\}]$. Recall that $M[\mathcal{C}_f(\mathcal{E})]$ is the state set of $\mathcal{S}_{\mathcal{E}}^M$ and hence we may deduce from Lemma A.2 that $\mathsf{post}^*_{\mathcal{S}_{\mathcal{E}}^M} = M[\mathcal{C}_f(\mathcal{E})]$.

Let us first prove assertion i). Since $M[\mathcal{C}_f(\mathcal{E})] \supseteq M_{\mathcal{T}_{\lhd}}$, it suffices to show that $M[\mathcal{C}_f(\mathcal{E})] \subseteq {\preceq}[M_{\mathcal{T}_{\lhd}}]$. Observe that every $C \in \mathcal{C}_f(\mathcal{E})$ with no $\lhd$-cutoff event is contained in $\mathcal{T}_{\lhd}$. It follows from Lemma F.1 that $M[\mathcal{C}_f(\mathcal{E})] \subseteq {\preceq}[M_{\mathcal{T}_{\lhd}}]$.

To prove assertion ii), consider a global label $a \in \Sigma_\gamma$. If a labels an event in $\mathcal{T}_{\lhd}(\mathcal{E}, M, \preceq)$ then we deduce from Lemma A.2 that a is quasi-live in $\mathcal{S}_{\mathcal{E}}^M$. To prove the converse, suppose that some execution in $\mathcal{S}_{\mathcal{E}}^M$ contains a transition labeled with a. We get that $s \xrightarrow{a} s'$ for some $s \in \mathsf{post}^*_{\mathcal{S}_{\mathcal{E}}^M}$. Since $s \in M[\mathcal{C}_f(\mathcal{E})]$, we obtain from Lemma F.1 that $M(C) \preceq s$ for some $C \in \mathcal{C}_f(\mathcal{E})$ containing no $\lhd$-cutoff event. Recall that $a \in \Sigma_\gamma$, and therefore reflexive compatibility of $(\mathcal{S}_{\mathcal{E}}^M, \succeq)$ implies that there is also a transition $M(C) \xrightarrow{a} t'$ for some $t' \preceq s'$. According to Lemma A.3 there exists an event $e \in E$ enabled at C such that $M(C \cup \{e\}) = s'$ and $l(e) = a$. Since C contains no $\lhd$-cutoff event, we obtain that $e \in \mathcal{T}_{\lhd}$ which concludes the proof. □

G From Forward Analysis to Backward Analysis in Well-Structured Transition Systems

We now show how to embed, in our forward partial-order analysis approach, standard backward analysis techniques (called "set saturation methods" in [FS01]) for well-structured transition systems.

We need a few additional notations in this section. Consider any labeled transition system $\mathcal{S} = (S, s^0, \Sigma, \rightarrow_{\mathcal{S}})$. The *one-step reachability relation* is the relation $\mathcal{R}_{\mathcal{S}}$ on S defined by $s \,\mathcal{R}_{\mathcal{S}}\, s'$ iff $s \xrightarrow{a} s'$ for some $a \in \Sigma$. We will use the following backward/forward (reachability) set transformers: for any subset $X \subseteq S$, we define $\mathsf{post}_{\mathcal{S}}(X) = \mathcal{R}_{\mathcal{S}}[X]$, $\mathsf{pre}_{\mathcal{S}}(X) = \mathcal{R}_{\mathcal{S}}^{-1}[X]$, $\mathsf{post}^*_{\mathcal{S}}(X) = (\mathcal{R}_{\mathcal{S}})^*[X]$, $\mathsf{pre}^*_{\mathcal{S}}(X) = (\mathcal{R}_{\mathcal{S}}^{-1})^*[X]$. Observe that the (previously defined) *reachability set* $\mathsf{post}^*_{\mathcal{S}}$ of $\mathcal{S}$ is equal to $\mathsf{post}^*_{\mathcal{S}}(s^0)$. For any label $a \in \Sigma$ and subset $X \subseteq S$, we define $\mathsf{pre}_{\mathcal{S}}(a, X) = \{s \in S \,/\, \exists\, s' \in X, s \xrightarrow{a} s'\}$. Remark that we have $\mathsf{pre}_{\mathcal{S}}(X) = \bigcup_{a \in \Sigma} \mathsf{pre}_{\mathcal{S}}(a, X)$ for every $X \subseteq S$.

Backward analysis for well-structured transition systems is performed by a classical fixpoint computation of $\mathsf{pre}^*({\preceq}[X])$ (applying an upward closure at each step), and requires a so-called finite pred-basis condition [AČJT00, FS01]. Consider any preordered LTS $(\mathcal{S}, \preceq)$ with $\mathcal{S} = (S, s^0, \Sigma, \rightarrow_{\mathcal{S}})$ and let us denote by $\mathbb{P}_f(S)$ the set of all finite subsets of S. A *finite pred-basis* for $(\mathcal{S}, \preceq)$ is any function *pb* from $\Sigma \times S$ to $\mathbb{P}_f(S)$ satisfying: $\bigcup_{b \in \Sigma} pb(b, s)$ is finite and ${\preceq}[pb(a, s)] = {\preceq}[\mathsf{pre}_{\mathcal{S}}(a, {\preceq}[\{s\}])]$ for every $a \in \Sigma$ and $s \in S$. Given any finite pred-basis *pb* for $(\mathcal{S}, \preceq)$, we define the *pb-reverse* of $(\mathcal{S}, \preceq, pb)$ as the labeled transition system $\mathcal{R}(\mathcal{S}, \preceq, pb) = (S, s^0, \Sigma, \rightarrow_{\mathcal{R}})$ where $\rightarrow_{\mathcal{R}}$ is defined by $s \xrightarrow{a}_{\mathcal{R}} s'$ iff $s' \in pb(a, s)$.

Lemma G.1. *For any preordered LTS* $(\mathcal{S}, \preceq)$ *with (standard) compatibility and finite pred-basis pb,* $\mathcal{R} = \mathcal{R}(\mathcal{S}, \preceq, pb)$ *is finitely-branching,* $(\mathcal{R}, \succeq)$ *has reflexive compatibility, and* $\mathcal{R}$ *satisfies* $\mathsf{pre}^*_{\mathcal{S}}(\preceq[\{s\}]) = \preceq[\mathsf{post}^*_{\mathcal{R}}(\{s\})]$ *for every state* s.

Proof. Consider any preordered LTS $(\mathcal{S}, \preceq)$ with (standard) compatibility and finite pred-basis pb, and let $\mathcal{R}$ shortly denote $\mathcal{R}(\mathcal{S}, \preceq, pb)$. Remark that $\mathcal{R}$ is finitely branching since $\bigcup_{a \in \Sigma} pb(a, s)$ is finite for every state s. Consider any transition $s \xrightarrow{a}_{\mathcal{R}} s'$ and let $t \preceq s$. It follows from $t \preceq s$ that $\preceq[pb(a, s)] = \preceq[\mathsf{pre}_{\mathcal{S}}(a, \preceq[\{s\}])] \subseteq \preceq[\mathsf{pre}_{\mathcal{S}}(a, \preceq[\{t\}])] = \preceq[pb(a, t)]$. Since $s' \in pb(a, s)$ we get that $s' \succeq t'$ for some $t' \in pb(a, t)$. Observe that $t \xrightarrow{a}_{\mathcal{R}} t'$ and hence $(\mathcal{R}, \succeq)$ has reflexive compatibility.

Now consider any state s and let us prove that $\mathsf{pre}^*_{\mathcal{S}}(\preceq[\{s\}]) = \preceq[\mathsf{post}^*_{\mathcal{R}}(\{s\})]$. Let us first prove inclusion $\subseteq$ and assume that $\pi = s_1 \xrightarrow{a_1}_{\mathcal{S}} s_2 \cdots s_k \xrightarrow{a_k}_{\mathcal{S}} s_{k+1}$ is a path in $\mathcal{S}$ such that $s_{k+1} \succeq s$. Let $t_{k+1} = s$. Observe that $s_k \in \mathsf{pre}_{\mathcal{S}}(a_k, \preceq[\{t_{k+1}\}]) \subseteq \preceq[pb(a_k, t_{k+1})]$. Since pb is a finite pred-basis for $(\mathcal{S}, \preceq)$, we get that $t_k \preceq s_k$ for some $t_k \in pb(a_k, t_{k+1})$ and hence $t_{k+1} \xrightarrow{a_k}_{\mathcal{R}} t_k$. By iterating this construction along the path π (from s_{k+1} to s_1) we get that there exists a path $s = t_{k+1} \xrightarrow{a_k}_{\mathcal{R}} t_k \xrightarrow{a_{k-1}}_{\mathcal{R}} t_{k-1} \cdots t_2 \xrightarrow{a_1}_{\mathcal{R}} t_1$ in $\mathcal{R}$ with $t_i \preceq s_i$, and in particular with $t_1 \preceq s_1$. Therefore $s_1 \in \preceq[\mathsf{post}^*_{\mathcal{R}}(\{s\})]$ which concludes the proof of this inclusion. Let us now prove inclusion $\supseteq$ and assume that $s = t_{k+1} \xrightarrow{a_k}_{\mathcal{R}} t_k \xrightarrow{a_{k-1}}_{\mathcal{R}} t_{k-1} \cdots t_2 \xrightarrow{a_1}_{\mathcal{R}} t_1$ is a path in $\mathcal{R}$ and let $s_1 \succeq t_1$. For every $1 \leq i \leq k$, we get that $t_i \in pb(a_i, t_{i+1}) \subseteq \preceq[\mathsf{pre}_{\mathcal{S}}(a_i, \preceq[\{t_{i+1}\}])]$ and hence there exist u_i, u_i' such that $t_i \succeq u_i \xrightarrow{a_i}_{\mathcal{S}} u_i' \succeq t_{i+1}$. We obtain from (standard) compatibility of $(\mathcal{S}, \preceq)$ that there exists $s_2 \succeq t_2, \ldots, s_{k+1} \succeq t_{k+1}$ such that $s_1 \xrightarrow{*}_{\mathcal{S}} s_2 \xrightarrow{*}_{\mathcal{S}} s_3 \cdots s_k \xrightarrow{*}_{\mathcal{S}} s_{k+1}$. Therefore $s_1 \in \mathsf{pre}^*_{\mathcal{S}}(\preceq[\{s\}])$ since $s_{k+1} \succeq t_{k+1} = s$. $\square$

We implicitly assumed that the partition of Σ into local labels and global labels is the same in $\mathcal{R}(\mathcal{S}, \preceq, pb)$ as in $\mathcal{S}$. It is clear from the proof of Lemma G.1 that reflexive compatibility of $(\mathcal{R}(\mathcal{S}, \preceq, pb), \succeq)$ does not depend on this partition (e.g. we may as well choose that all labels are global in $\mathcal{R}$).

The previous lemma combined with proposition 3.7 allows us to reduce backward analysis to forward analysis: to compute $\mathsf{pre}^*_{\mathcal{S}}(\preceq[\{s\}])$ in a well-preordered LTS $(\mathcal{S}, \preceq)$ with (standard) compatibility and finite pred-basis pb, it is sufficient to build the finite truncation $\mathcal{T}_{\triangleleft}$ of a complete marked LES $\mathcal{E}$ for $\mathcal{R}(\mathcal{S}, \preceq, pb)$.