

# Computation of summaries using net unfoldings

Javier Esparza<sup>1</sup>, Loïc Jezequel<sup>2</sup>, and Stefan Schwoon<sup>3</sup>

- 1 Institut für Informatik  
Technische Universität München, Germany  
esparza@in.tum.de
- 2 ENS Cachan Bretagne  
Rennes, France  
jezequel@model.in.tum.de
- 3 LSV, ENS Cachan & CNRS  
INRIA Saclay, France  
schwoon@lsv.ens-cachan.fr

---

## Abstract

We study the following *summarization problem*: given a parallel composition  $\mathbf{A} = \mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$  of labelled transition systems communicating with the environment through a distinguished component  $\mathcal{A}_i$ , efficiently compute a *summary*  $\mathcal{S}_i$  such that  $\mathbf{E} \parallel \mathbf{A}$  and  $\mathbf{E} \parallel \mathcal{S}_i$  are trace-equivalent for every environment  $\mathbf{E}$ . While  $\mathcal{S}_i$  can be computed using elementary automata theory, the resulting algorithm suffers from the state-explosion problem. We present a new, simple but subtle algorithm based on net unfoldings, a partial-order semantics, give some experimental results using an implementation on top of MOLE, and show that our algorithm can handle *divergences* and compute *weighted summaries* with minor modifications.

**1998 ACM Subject Classification** D.2.2 Design Tools and Techniques, F.1.2 Modes of Computation

**Keywords and phrases** Net unfoldings, Concurrent systems, Petri nets

**Digital Object Identifier** 10.4230/LIPIcs.xxx.yyy.p

## 1 Introduction

We address a fundamental problem in automatic compositional verification. Consider a parallel composition  $\mathbf{A} = \mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$  of processes, modelled as labelled transition systems, which is itself part of a larger system  $\mathbf{E} \parallel \mathbf{A}$  for some environment  $\mathbf{E}$ . Assume that  $\mathcal{A}_i$  is the interface of  $\mathbf{A}$  with the environment, i.e.,  $\mathbf{A}$  communicates with the outer world only through actions of  $\mathcal{A}_i$ . The task consists in computing a new interface  $\mathcal{S}_i$  with the same set of actions as  $\mathcal{A}_i$  such that  $\mathbf{E} \parallel \mathbf{A}$  and  $\mathbf{E} \parallel \mathcal{S}_i$  have the same behaviour. In other words, the environment  $E$  cannot distinguish between  $\mathbf{A}$  and  $\mathcal{S}_i$ . Since  $\mathcal{S}_i$  usually has a much smaller state space than  $\mathbf{A}$  (making  $\mathbf{E} \parallel \mathbf{A}$  easier to analyse) we call it a *summary*.

We study the problem in a CSP-like setting [12]: parallel composition is by rendez-vous, and the behaviour of a transition system is given by its trace semantics.

It is easy to compute  $\mathcal{S}_i$  using elementary automata theory: we first compute the transition system of  $\mathbf{A}$ , whose states are tuples  $(s_1, \dots, s_n)$ , where  $s_i$  is a state of  $\mathcal{A}_i$ . Then we hide all actions except those of the interface, i.e., we replace them by  $\varepsilon$ -transitions ( $\tau$ -transitions in CSP terminology). We can then eliminate all  $\varepsilon$ -transitions using standard algorithms, and, if desired, compute the minimal summary by applying e.g. Hopcroft's algorithm. The problem of this approach is the state-space explosion: the number of states of  $\mathbf{A}$  can grow exponentially in the number of sequential components. While this is unavoidable in the worst



© J. Esparza, L. Jezequel and S. Schwoon;  
licensed under Creative Commons License CC-BY

Conference title on which this volume is based on.

Editors: Billy Editor, Bill Editors; pp. 1–22



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

case (deciding whether  $\mathcal{S}_i$  has an empty set of traces is a PSPACE-complete problem, and the minimal summary  $\mathcal{S}_i$  may be exponentially larger than  $\mathcal{A}_1, \dots, \mathcal{A}_n$  in the worst case, see e.g. [10]) the combinatorial explosion happens already in trivial cases: if the components  $\mathcal{A}_1, \dots, \mathcal{A}_n$  do not communicate at all, we can obviously take  $\mathcal{S}_i = \mathcal{A}_i$ , but the algorithm we have just described will need exponential time and space.

We present a technique to palliate this problem based on net unfoldings (see e.g. [4]). Net unfoldings are a partial-order semantics for concurrent systems, closely related to event structures [21], that provides very compact representations of the state space for systems with a high degree of concurrency. Intuitively, an unfolding is the extension to parallel compositions of the notion of unfolding a transition system into a tree. The unfolding is usually infinite. We show how to algorithmically construct a finite prefix of it from which the summary can be easily extracted. The algorithm can be easily implemented re-using many components of existing unfolders like PUNF<sup>1</sup> and MOLE<sup>2</sup>. However, its correctness proof is surprisingly subtle. This proof is the main contribution of the paper, but we also evaluate the algorithm on some classical benchmarks [2].

**Related work.** The summarization problem has been extensively studied in an interleaving setting (see e.g. [9, 20, 22]), in which one first constructs the transition system of  $\mathbf{A}$  and then reduces it. We study it in a partial-order setting.

Net unfoldings, and in general partial-order semantics, have been used to solve many analysis problems: deadlock [17, 14], reachability and model-checking questions [5, 3, 13, 4, 1], diagnosis [6], and other specific applications [16, 11]. To the best of our knowledge we are the first to explicitly study the summarization problem.

Our problem can be solved with the help of Zielonka's algorithm [23, 18, 8], which yields an asynchronous automaton trace-equivalent to  $\mathbf{A}$ . The projection of this automaton onto the alphabet of  $\mathcal{A}_i$  yields a summary  $\mathcal{S}_i$ . However, Zielonka's algorithm is notoriously complicated and, contrary to our algorithm, requires to store much additional information for each event [18].

In [7], the complete tuple  $\mathcal{S}_1, \dots, \mathcal{S}_n$  is computed by means of an iterative message-passing algorithm that transfers information between components until a fixed point is reached. However, the fixed point is only guaranteed to be reached when the communication graph is acyclic.

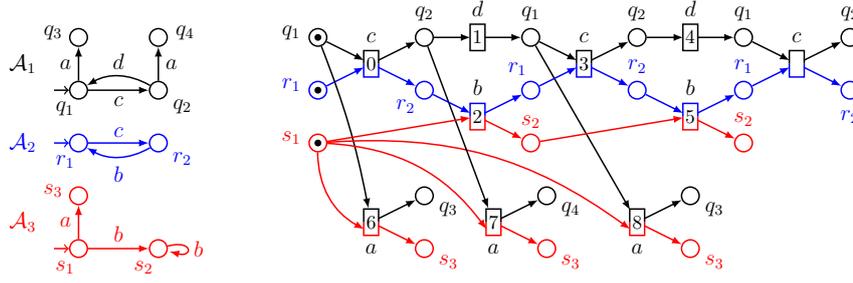
## 2 Preliminaries

**Transition systems.** A *labelled transition system* (LTS) is a tuple  $\mathcal{A} = (\Sigma, S, T, \lambda, s^0)$  where  $\Sigma$  is a set of *actions*,  $S$  is a set of *states*,  $T \subseteq S \times S$  is a set of *transitions*,  $\lambda: T \rightarrow \Sigma$  is a *labelling function*, and  $s^0 \in S$  is an *initial state*. An *a-transition* is a transition labelled by  $a$ . A (finite or infinite) action sequence  $\sigma = a_1 a_2 a_3 \dots \in \Sigma^* \cup \Sigma^\omega$  is a *trace* of  $\mathcal{A}$  if there is a (finite or infinite) sequence  $s_0 s_1 s_2 \dots$  of states such that  $s_0 = s^0$ ,  $t_i = (s_{i-1}, s_i) \in T$  and  $\lambda(t_i) = a_i$  for every  $i$ . The path  $s_0 \dots s_n$  is a *realization* of  $\sigma$ . The set of traces of  $\mathcal{A}$  is denoted by  $Tr(\mathcal{A})$ . Figure 1 shows three transition systems.

Let  $\mathcal{A}_1, \dots, \mathcal{A}_n$  be LTSs where  $\mathcal{A}_i = (\Sigma_i, S_i, T_i, \lambda_i, s_i^0)$ . The *parallel composition*  $\mathbf{A} = \mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$  is the LTS defined as follows. The set of actions is  $\Sigma = \Sigma_1 \cup \dots \cup \Sigma_n$ . The states, called *global states*, are the tuples  $\mathbf{s} = (s_1, \dots, s_n)$  such that  $s_i \in S_i$  for every  $i \in \{1..n\}$ . The *initial global state* is  $\mathbf{s}^0 = (s_1^0, \dots, s_n^0)$ . The transitions, called *global transitions*, are

<sup>1</sup> [homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/](http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/)

<sup>2</sup> <http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/>



■ **Figure 1** Three labeled transition systems (left) and a branching process (right)

the tuples  $\mathbf{t} = (t_1, \dots, t_n) \in (T_1 \cup \{\star\}) \times \dots \times (T_n \cup \{\star\}) \setminus \{(\star, \dots, \star)\}$  such that there is an action  $a \in \Sigma$  satisfying for every  $i \in \{1..n\}$ : if  $a \in \Sigma_i$ , then  $t_i$  is an  $a$ -transition of  $T_i$ , otherwise  $t_i = \star$ ; the label of  $\mathbf{t}$  is the action  $a$ . If  $t_i \neq \star$  we say that  $\mathcal{A}_i$  *participates* in  $\mathbf{t}$ . It is easy to see that  $\sigma \in \Sigma^* \cup \Sigma^\omega$  is a trace of  $\mathbf{A}$  iff for every  $i \in \{1..n\}$  the projection of  $\sigma$  on  $\Sigma_i$ , denoted by  $\sigma|_{\Sigma_i}$ , is a trace of  $\mathcal{A}_i$ .

**Petri nets.** A *labelled net* is a tuple  $(\Sigma, P, T, F, \lambda)$  where  $\Sigma$  is a set of *actions*,  $P$  and  $T$  are disjoint sets of *places* and *transitions* (jointly called *nodes*),  $F \subseteq (P \times T) \cup (T \times P)$  is a set of *arcs*, and  $\lambda: P \cup T \rightarrow \Sigma$  is a *labelling function*. For  $x \in P \cup T$  we denote by  $\bullet x = \{y \mid (y, x) \in F\}$  and  $x^\bullet = \{y \mid (x, y) \in F\}$  the sets of *inputs* and *outputs* of  $x$ , respectively. A set  $M$  of places is called a *marking*. A *labelled Petri net* is a tuple  $\mathcal{N} = (\Sigma, P, T, F, \lambda, M_0)$  where  $(\Sigma, P, T, F, \lambda)$  is a net and  $M_0 \subseteq P$  is the *initial marking*. A marking  $M$  *enables* a transition  $t \in T$  if  $\bullet t \subseteq M$ . In this case  $t$  can *occur* or *fire*, leading to the new marking  $M' = (M \setminus \bullet t) \cup t^\bullet$ . An *occurrence sequence* is a (finite or infinite) sequence of transitions that can occur from  $M_0$  in the order specified by the sequence. A *trace* is the sequence of labels of an occurrence sequence. The set of traces of  $\mathcal{N}$  is denoted by  $Tr(\mathcal{N})$ .

**Branching processes.** The finite *branching processes* of  $\mathbf{A} = \mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$  are labelled Petri nets whose places are labelled with states of  $\mathcal{A}_1, \dots, \mathcal{A}_n$ , and whose transitions are labelled with global transitions of  $\mathbf{A}$ . (Since global transitions are labelled with actions, each event is also implicitly labelled with an action.) Following tradition, we call the places and transitions of these nets *conditions* and *events*, respectively. We say that a marking  $M$  of these nets *enables* a global transition  $\mathbf{t}$  of  $\mathbf{A}$  if for every state  $s \in \bullet \mathbf{t}$  some condition of  $M$  is labelled by  $s$ . The set of *finite branching processes* of  $\mathbf{A}$  is defined inductively as follows:

1. A labelled Petri net with conditions  $b_1^0, \dots, b_n^0$  labelled by  $s_1^0, \dots, s_n^0$ , no events, and with initial marking  $\{b_1^0, \dots, b_n^0\}$ , is a branching process of  $\mathbf{A}$ .
2. Let  $\mathcal{N}$  be a branching process of  $\mathbf{A}$  such that some reachable marking of  $\mathcal{N}$  enables some global transition  $\mathbf{t}$ . Let  $M$  be the subset of conditions of the marking labelled by  $\bullet \mathbf{t}$ . If  $\mathcal{N}$  has no event labelled by  $\mathbf{t}$  with  $M$  as input set, then the Petri net obtained by adding to  $\mathcal{N}$ : a new event  $e$ , labelled by  $\mathbf{t}$ ; a new condition for every state  $s$  of  $\mathbf{t}^\bullet$ , labelled by  $s$ ; new arcs leading from each condition of  $M$  to  $e$ , and from  $e$  to each of the new conditions, is also a branching process of  $\mathbf{A}$ .

Figure 1 shows on the right a branching process of the parallel composition of the LTSs on the left. Events are labelled with their corresponding actions.

The set of all branching processes of a net, finite and infinite, is defined by closing the finite branching processes under countable unions (after a suitable renaming of conditions and events) [4]. In particular, the union of all finite branching processes yields the *unfolding* of the net, which intuitively corresponds to the result of exhaustively adding all extensions in the definition above.

A *trace* of a branching process  $\mathcal{N}$  is the sequence of action labels of an occurrence sequence of events of  $\mathcal{N}$ . In Figure 1, firing the events on the top half of the process yields any of the traces  $cbdcdb$ ,  $cdbcb$ ,  $cbdcdb$ , or  $cbdcdb$ . The sets of traces of  $\mathbf{A}$  and of its unfolding coincide.

Let  $x, y$  be nodes of a branching process. We say that  $x$  is a *causal predecessor* of  $y$ , denoted by  $x < y$ , if there is a non-empty path of arcs from  $x$  to  $y$ ; further,  $x \leq y$  denotes that either  $x < y$  or  $x = y$ . If  $x \leq y$  or  $x \geq y$ , then  $x$  and  $y$  are *causally related*. We say that  $x$  and  $y$  are *in conflict*, denoted by  $x \# y$ , if there is a condition  $z$  (different from  $x$  and  $y$ ) from which one can reach both  $x$  and  $y$ , exiting  $z$  by different arcs. Finally,  $x$  and  $y$  are *concurrent* if they are neither causally related nor in conflict.

A set of events  $E$  is a *configuration* if it is *causally closed* (that is, if  $e \in E$  and  $e' < e$  then  $e' \in E$ ) and *conflict-free* (that is, for every  $e, e' \in E$ ,  $e$  and  $e'$  are not in conflict). The *past* of an event  $e$ , denoted by  $[e]$ , is the set of events  $e'$  such that  $e' \leq e$  (so it is a configuration). For any event  $e$ , we denote by  $M(e)$  the unique marking reached by any occurrence sequence that fires exactly the events of  $[e]$ . Notice that, for each component  $\mathcal{A}_i$  of  $\mathbf{A}$ ,  $M(e)$  contains exactly one condition labelled by a state of  $\mathcal{A}_i$ . We denote this condition by  $M(e)_i$ . We write  $\mathbf{St}(e) = \{ \lambda(x) \mid x \in M(e) \}$  and call it the *global state reached by  $e$* .

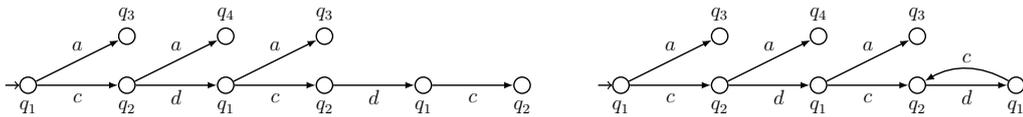
### 3 The Summary Problem

Let  $\mathbf{A} = \mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$  be a parallel composition with a distinguished component  $\mathcal{A}_i$ , called the *interface*. An *environment* of  $\mathbf{A}$  is any LTS  $\mathbf{E}$  (possibly a parallel composition) that only communicates with  $\mathbf{A}$  through the interface, i.e.,  $\Sigma_{\mathbf{E}} \cap (\Sigma_1 \cup \dots \cup \Sigma_n) = \Sigma_{\mathbf{E}} \cap \Sigma_i$ . We wish to compute a *summary*  $\mathcal{S}_i$ , i.e., an LTS with the same actions as  $\mathcal{A}_i$  such that  $Tr(\mathbf{E} \parallel \mathbf{A})|_{\Sigma_{\mathbf{E}}} = Tr(\mathbf{E} \parallel \mathcal{S}_i)|_{\Sigma_{\mathbf{E}}}$  for every environment  $\mathbf{E}$ , where  $X|_{\Sigma}$  denotes the projection of the traces of  $X$  onto  $\Sigma$ . It is well known (and follows easily from the definitions) that this holds iff  $Tr(\mathcal{S}_i) = Tr(\mathbf{A})|_{\Sigma_i}$  [12]. We therefore address the following problem:

► **Definition 1** (Summary problem). Given LTSs  $\mathcal{A}_1, \dots, \mathcal{A}_n$  with interface  $\mathcal{A}_i$ , compute an LTS  $\mathcal{S}_i$  satisfying  $Tr(\mathcal{S}_i) = Tr(\mathbf{A})|_{\Sigma_i}$ , where  $\mathbf{A} = \mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$ .

The problem can be solved by computing the LTS  $\mathbf{A}$ , but the size of  $\mathbf{A}$  can be exponential in  $\mathcal{A}_1, \dots, \mathcal{A}_n$ . So we investigate an unfolding approach.

The *interface projection*  $\mathcal{N}_i$  of a branching process  $\mathcal{N}$  of  $\mathbf{A}$  onto  $\mathcal{A}_i$  is the following labelled subnet of  $\mathcal{N}$ : (1) the conditions of  $\mathcal{N}_i$  are the conditions of  $\mathcal{N}$  with labels in  $\mathcal{S}_i$ ; (2) the events of  $\mathcal{N}_i$  are the events of  $\mathcal{N}$  where  $\mathcal{A}_i$  participates; (3)  $(x, y)$  is an arc  $\mathcal{N}_i$  iff it is an arc of  $\mathcal{N}$  and  $(x, y)$  are nodes of  $\mathcal{N}_i$ . Obviously, every event of  $\mathcal{N}_i$  has exactly one input and one output condition, and  $\mathcal{N}_i$  can therefore be seen as an LTS; abusing language, sometimes we speak of the LTS  $\mathcal{N}_i$ . The interface projection  $\mathcal{N}_1$  for the branching process of Figure 1 is the subnet given by the black conditions and their input and output events, and its LTS representation is shown in the left of Figure 2.



■ **Figure 2** Projection of the branching process of Figure 1 on  $\mathcal{A}_1$  (left) and a folding (right)

The projection  $\mathcal{U}_i$  of the full unfolding of  $\mathbf{A}$  onto  $\mathcal{A}_i$  clearly satisfies  $Tr(\mathcal{U}_i) = Tr(\mathbf{A})|_{\Sigma_i}$ ; however,  $\mathcal{U}_i$  can be an infinite transition system. In the rest of the paper we show how to compute a *finite* branching process  $\mathcal{N}$  and an equivalence relation  $\equiv$  between the conditions

of  $\mathcal{N}_i$  such that the result of *folding*  $\mathcal{N}_i$  into a finite transition system by merging the conditions of each equivalence class yields the desired  $\mathcal{S}_i$ . The *folding* of  $\mathcal{N}_i$  with respect to an equivalence relation  $\equiv$  on the conditions of  $\mathcal{N}_i$  is standardly defined: its states are the equivalence classes of  $\equiv$ , and every transition  $(s, s')$  of  $\mathcal{N}_i$  yields a transition  $([s]_{\equiv}, [s']_{\equiv})$  of the folding. Figure 2 shows on the right the result of folding the LTS on the left when the only equivalence class with more than one member is formed by the two rightmost states labelled by  $q_2$ .

We construct  $\mathcal{N}$  by starting with the branching processes without events and iteratively adding events according to the definition of branching processes. Some events are marked as *cut-offs* [4]. An event  $e$  added to  $\mathcal{N}$  becomes a cut-off if  $\mathcal{N}$  already contains an  $e'$ , called the *companion* of  $e$ , satisfying a certain, yet to be specified *cut-off condition*. Events with cut-offs in their past cannot be added. The algorithm terminates when no more events can be added. The equivalence relation  $\equiv$  is determined by the *interface cut-offs*: the cut-offs labelled with interface actions. If an interface cut-off  $e$  has companion  $e'$ , then we set  $M(e)_i \equiv M(e')_i$ . Algorithm 1 is pseudocode for the unfolding, where  $Ext(\mathcal{N}, co)$  denotes the *possible extensions*: the events which can be added to  $\mathcal{N}$  without events from  $co$  in their past.

---

**Algorithm 1** Unfolding procedure for a product **A**.

---

```

let  $\mathcal{N}$  be the unique branching process of A without events and let  $co = \emptyset$ 
While  $Ext(\mathcal{N}, co) \neq \emptyset$  do
  choose  $e$  in  $Ext(\mathcal{N}, co)$  and extend  $\mathcal{N}$  with  $e$ 
  If  $e$  is a cut-off event then let  $co = co \cup \{e\}$ 
  For every  $e \in co$  with companion  $e'$  do merge  $[M(e)_i]_{\equiv}$  and  $[M(e')_i]_{\equiv}$ 

```

---

Notice that the algorithm is nondeterministic: the order in which events are added is not fixed (other than the fact that this order must necessarily respect causal relations). Our task consists in finding a definition of cut-offs such that the LTS  $\mathcal{S}_i$  delivered by the algorithm with this definition is a correct solution to the summary problem. Several papers have adressed the problem of defining cut-offs such that the branching process delivered by the algorithm contains all global states of the system (see [4] and the references therein). Appendix A.1 shows that these approaches do not “unfold enough”.

## 4 Two Attempts

The right definition of cut-offs turns out to be remarkably subtle, and so we approach it in a series of steps.

### 4.1 First attempt

In the following we shall call events in which  $\mathcal{A}_i$  participates *i-events* for short; analogously, we call *i-conditions* the conditions labelled by states of  $\mathcal{A}_i$ .

The simplest idea is to declare an *i-event*  $e$  a cut-off if the branching process already contains another *i-event*  $e'$  such that  $\mathbf{St}(e) = \mathbf{St}(e')$ . Intuitively, the behaviours of the interface the configurations  $[e]$  and  $[e']$  is identical, and so we only explore the future of  $[e']$ .

**Cut-off definition 1.** An event  $e$  is a cut-off event if it is an *i-event* and  $\mathcal{N}$  contains an *i-event*  $e'$  such that  $\mathbf{St}(e) = \mathbf{St}(e')$ .

It is not difficult to show that this definition is correct for *non-divergent* systems.

► **Definition 2.** A parallel composition  $\mathbf{A}$  with interface  $\mathcal{A}_i$  is *divergent* if some infinite trace of  $\mathbf{A}$  contains only finitely many occurrences of actions of  $\Sigma_i$ .

► **Theorem 3.** Let  $\mathbf{A}$  be non-divergent. The instance of Algorithm 1 with cut-off definition 1 terminates with a finite branching process  $\mathcal{N}$ , and the folding  $\mathcal{S}_i$  of  $\mathcal{N}_i$  is a summary of  $\mathbf{A}$ .

The proof of this theorem is given in Appendix A.2

The system of Figure 1 is non-divergent. Algorithm 1 computes the branching process on the right of Figure 1. The only cut-off is the dashed event with companion 3. The folding is shown in Figure 2 (right) and is a correct summary.

However, cut-off definition 1 *never* works if  $\mathbf{A}$  is divergent because the unfolding procedure does not terminate. Indeed, if the system has divergent traces then we can easily construct an infinite firing sequence of the unfolding such that none of the finitely many  $i$ -events in the sequence is a cut-off. Since no other events can be cut-offs, Algorithm 1 adds all events of the sequence. This occurs for instance for the system of Figure 3 with interface  $\mathcal{A}_1$ , where the occurrence sequence of the unfolding for the trace  $i(fcd)^\omega$  contains no cut-off.

## 4.2 Second attempt

To ensure termination for divergent systems, we extend the definition of cut-off. For this, we define for each event  $e$  its  *$i$ -predecessor*. Intuitively, the  $i$ -predecessor of an event  $e$  is the last condition that  $e$  “knows” has been reached by the interface.

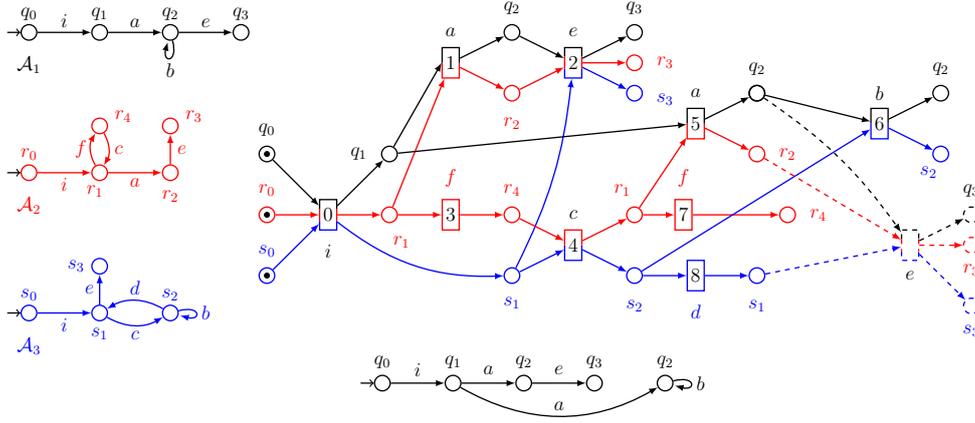
► **Definition 4.** Let  $e$  be an event. The  *$i$ -predecessor* of  $e$ , denoted by  $ip(e)$ , is the condition  $M(e)_i$ .

Assume now that two events  $e_1 < e_2$ , neither of them interface event, satisfy  $ip(e_1) = ip(e_2)$  and  $\mathbf{St}(e_1) = \mathbf{St}(e_2)$ . Then any occurrence sequence  $\sigma$  that executes the events of the set  $[e_2] \setminus [e_1]$  leads from a marking to itself and contains no interface events. So  $\sigma$  can be repeated infinitely often, leading to an infinite trace with only finitely many interface actions. It is therefore plausible to mark  $e_2$  as cut-off event, in order to avoid this infinite repetition.

**Cut-off definition 2.** An event  $e$  is a cut-off if

- (1)  $e$  is an  $i$ -event, and  $\mathcal{N}$  contains an  $i$ -event  $e'$  with  $\mathbf{St}(e) = \mathbf{St}(e')$ , or
- (2)  $e$  is not an  $i$ -event, and  $\mathcal{N}$  contains an event  $e' < e$  such that  $\mathbf{St}(e) = \mathbf{St}(e')$  and  $ip(e) = ip(e')$ .

A first contribution of our paper is an example showing that this natural definition does not work: the algorithm always terminates but can yield a wrong result. Consider the parallel composition at the left of Figure 3, with interface  $\mathcal{A}_1$ . It is easy to see that  $Tr(\mathcal{A})|_{\Sigma_1} = Tr(\mathcal{A}_1) = iab^*e$ . For any strategy the algorithm generates the branching process  $\mathcal{N}$  at the top right of the figure.  $\mathcal{N}$  has two cut-off events: the interface event 6, which is of type (1), and event 8, a non-interface event, of type (2). Event 6 has 5 as companion, with  $\mathbf{St}(5) = \{q_2, r_2, s_2\}$ . Event 8 has 0 as companion, with  $\mathbf{St}(0) = \{q_1, r_1, s_1\} = \mathbf{St}(8)$ ; moreover,  $0 < 8$  and  $ip(0) = ip(8)$ . The folding of  $\mathcal{N}_1$  is shown at the bottom right of the figure. It is clearly not trace-equivalent to  $\mathcal{A}_1$  because it “misses” the trace  $iabe$ . The dashed event at the bottom right, which would correct this, is not added by the algorithm because it is a successor of 8.



■ **Figure 3** Cut-off definition 2 produces an incorrect result on  $\mathbf{A} = \mathcal{A}_1 \parallel \mathcal{A}_2 \parallel \mathcal{A}_3$

## 5 The Solution

Intuitively, the reason for the failure of our third attempt on the example of Figure 3 is that  $\mathcal{A}_1$  can only execute  $iabe$  if  $\mathcal{A}_2$  and  $\mathcal{A}_3$  execute  $ifcd$  first. However, when the algorithm observes that the markings before and after the execution of  $ifcd$  are identical, it declares 8 a cut-off event, and so it cannot “use” it to construct event  $e$ . So, on the one hand, 8 should not be a cut-off event. But, on the other hand, *some* event of the trace  $i(fcd)^\omega$  must be declared cut-off, otherwise the algorithm does not terminate.

The way out of this dilemma is to introduce *cut-off candidates*. If an event is declared a cut-off candidate, the algorithm does not add any of its successors, just as with regular cut-offs. However, cut-off candidates may stop being candidates if the addition of a new event *frees* them. (So, an event is a cut-off candidate *with respect to the current branching process*.) A generic unfolding procedure using these ideas is given in Algorithm 2, where  $Ext(\mathcal{N}, co, coc)$  denotes the possible extensions of  $\mathcal{N}$  that do not have any event of  $co$  or  $coc$  in their past. Assuming suitable definitions of cut-off candidates and freeing, the algorithm would, in our example, declare event 8 a cut-off candidate, momentarily stop adding any of its successors, but later free event 8 when event 5 is discovered.

---

### Algorithm 2 Unfolding procedure for a product $\mathbf{A}$ .

---

let  $\mathcal{N}$  be the unique branching process of  $\mathbf{A}$  without events; let  $co = \emptyset$  and  $coc = \emptyset$

**While**  $Ext(\mathcal{N}, co, coc) \neq \emptyset$  **do**

    choose  $e$  in  $Ext(\mathcal{N}, co, coc)$  according to the search strategy

**If**  $e$  is a cut-off event **then** let  $co = co \cup \{e\}$

**Elseif**  $e$  is a cut-off candidate of  $\mathcal{N}$  **then** let  $coc = coc \cup \{e\}$

**Else for every**  $e' \in coc$  **do**

**If**  $e$  frees  $e'$  **then**  $coc = coc \setminus \{e'\}$

    extend  $\mathcal{N}$  with  $e$

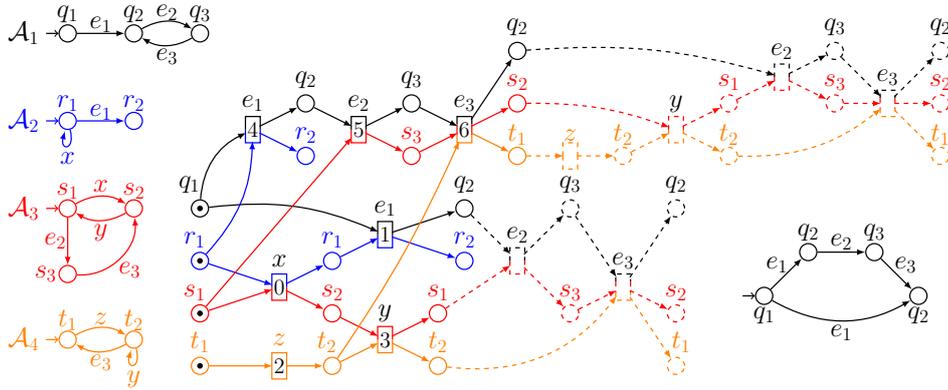
**For every**  $e \in co$  with companion  $e'$  **do** merge  $[M(e)_i]_{\equiv}$  and  $[M(e')_i]_{\equiv}$

---

The main contribution of our paper is the definition of a correct notion of cut-off candidate for the projection problem. We shall declare event  $e$  a cut-off candidate if  $e$  is not an interface event, and  $\mathcal{N}$  contains a companion  $e' < e$  such that  $\mathbf{St}(e') = \mathbf{St}(e)$ ,  $ip(e) = ip(e')$ , and, additionally, no interface event  $e''$  of  $\mathcal{N}$  is concurrent with  $e$  without being concurrent with

$e'$ . As long as this condition holds, the successors of  $e$  are put “on hold”. In the example of Figure 3, if the algorithm first adds events 0, 3, 4, and 8, then event 8 becomes a cut-off candidate with 0 as companion. However, the addition of the interface event 5 frees event 8, because 5 is concurrent with 8 and not with 0.

However, we are not completely done yet. The parallel composition at the left of Figure 4 gives an example in which even with this notion of cut-off candidate the result is still wrong.  $\mathcal{A}_1$  is the interface. One branching process is represented at the top right of the figure. Event 3 (concurrent with 1) is a cut-off candidate with 2 (concurrent with 1, 4, and 5) as companion. This prevents the lower dashed part of the net to be added. Event 6 is cut-off with 1 as companion. This prevents the upper dashed part of the net to be added. The refolding obtained then (bottom right) does not contain the word  $e_1e_2e_3e_2$ .



■ **Figure 4** An example illustrating the use of strong causality

If we wish a correct algorithm for all strategies, we need a final touch: replace the condition  $e' < e$  by  $e' \ll e$ , where  $\ll$  is the *strong causal relation*:

► **Definition 5.** Event  $e'$  is a *strong cause* of event  $e$ , denoted by  $e' \ll e$ , if  $e' < e$  and  $b' < b$  for every  $b \in M(e) \setminus M(e')$ ,  $b' \in M(e') \setminus M(e)$ .

Using this definition, event 3 is no longer a cut-off candidate in the branching process of Figure 4 as it is not in strong causal relation with its companion 2 (because the  $t_2$ -labelled condition just after 2 belongs to  $M(2) \setminus M(3)$  and is not causally related with the  $r_1$ -labelled condition just after 0 which belongs to  $M(3) \setminus M(2)$ ).

We are now in a position to provide adequate definitions for Algorithm 2.

► **Definition 6** (Cut-off and cut-off candidate). Let  $Ico_{\mathcal{N}}(e)$  denote the set of non cut-off interface events of  $\mathcal{N}$  that are concurrent with  $e$ . An event  $e$

- is a cut-off if it is an  $i$ -event, and  $\mathcal{N}$  contains an  $i$ -event  $e'$  such that  $\mathbf{St}(e) = \mathbf{St}(e')$ .
- is a cut-off candidate of  $\mathcal{N}$  if it is not an  $i$ -event, and  $\mathcal{N}$  contains  $e' \ll e$  such that  $\mathbf{St}(e) = \mathbf{St}(e')$ ,  $ip(e') = ip(e)$ , and  $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$ .
- frees a cut-off candidate  $e_c$  of  $\mathcal{N}$  if  $e_c$  is not a cut-off candidate of the branching process obtained by adding  $e$  to  $\mathcal{N}$ .

► **Theorem 7.** Let  $\mathbf{A} = \mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$  with interface  $\mathcal{A}_i$ . The instance of Algorithm 2 given by Definition 6 terminates and returns a branching process  $\mathcal{N}$  such that the folding  $\mathcal{S}_i$  of  $\mathcal{N}_i$  is a summary of  $\mathbf{A}$ .

The proof of this Theorem is involved. It is given in the Appendix A.3. We sketch the main ideas. Termination follows from a lemma showing that every infinite chain of causally related events contains an infinite subchain of strongly causally related events. The equality  $Tr(\mathcal{S}_i) = Tr(\mathbf{A})|_{\Sigma_i}$  is proved in two parts.  $Tr(\mathcal{S}_i) \subseteq Tr(\mathbf{A})|_{\Sigma_i}$  follows easily from the definitions. The proof of  $Tr(\mathcal{S}_i) \supseteq Tr(\mathbf{A})|_{\Sigma_i}$  is more involved. For every trace of  $\mathbf{A}$  we identify a *strongly succinct* occurrence sequence in the unfolding with this trace as projection. Intuitively, in such a sequence, interface events occur as early as possible, and the number of non-interface events occurring between them is minimal. The main point in the proof is to show that every cut-off in strongly succinct sequences is necessarily an interface event, which allows one to conclude the proof as in the non-divergent case. This is proved by contradiction: If  $e$  is a cut-off candidate with companion  $e'$ , we show that (1)  $e$  and  $e'$  are located between the same two interface events (this uses  $Ico_N(e) \subseteq Ico_N(e')$ ), (2) there is no  $i$ -event in  $[e] \setminus [e']$ , and (3) every event of  $[e] \setminus [e']$  is also located between the same two interface events (this is ensured by  $e' \ll e$ ). The events from  $[e] \setminus [e']$  can then be removed from the sequence, contradicting the definition of strongly succinct sequence.

## 6 Implementation and Experiments

As an illustration of the previous results, we report in this section on an implementation of Algorithm 2 as a modification of the existing unfolding tool MOLE. All programs and data used are publicly available.<sup>3</sup> Many components of MOLE could be re-used. The main work consisted in determining cut-off candidates and the “freeing” condition of Definition 6. This required two main algorithmic additions discussed in detail in Appendix B: (i) an efficient traversal of  $[e]$ , for a given event  $e$  that allows to determine the conditions for cut-off candidates; (ii) computing  $Ico_N(e)$  for an event  $e$ . Both additions could be obtained by extending existing components of the tool. While the additions were not always trivial, they could be obtained with small additional overhead.

We tested our implementation on well-known benchmarks used widely in the unfolding literature, see e.g. [2, 5, 15]. The input is the set of components  $\mathcal{A}_1, \dots, \mathcal{A}_n$ , which are converted into an equivalent Petri net. For each example, we report the number of events (including cutoffs) in the prefix. Notice that this prefix is computed in less than one second in most cases (more detailed experimental results are given in Appendix C). We also report the number of reachable markings (taken from [19] where available, and computed combinatorially for DPSYN). Comparing this with the number of events gives a rough indication on how long a ‘trivial’ algorithm that computes the entire reachability graph (cf. Introduction) would take compared to our approach.

Test case	Events	Markings
CYCLIC(6)	426	639
CYCLIC(9)	3347	7423
CYCLIC(12)	26652	74264
CYCLICS(6)	303	639
CYCLICS(9)	2328	7423
CYCLICS(12)	18464	74264
DAC(9)	86	1790
DAC(12)	134	14334
DAC(15)	191	114686
DP(6)	935	729
DP(8)	5121	6555
DP(10)	31031	48897
DPD(4)	2373	601
DPD(5)	23789	3489
DPD(6)	245013	19861
DPSYN(10)	176	123
DPSYN(20)	701	15127
DPSYN(30)	1576	1860498
RING(5)	511	1290
RING(7)	3139	17000
RING(9)	16799	211528

■ **Table 1** Experimental results

<sup>3</sup> [http://www.lsv.ens-cachan.fr/~sim\\$schwoon/tools/mole/summaries.tar.gz](http://www.lsv.ens-cachan.fr/~sim$schwoon/tools/mole/summaries.tar.gz)

The experiments are summarized in Table 1. We used the following families of examples: the CYCLIC and CYCLICS families are a model of Milner’s cyclic scheduler with  $n$  consumers and  $n$  schedulers; in one case we compute the folding for a consumer, in the other for a scheduler. The DAC family represents a divide-and-conquer computation. RING is a mutual-exclusion protocol on a token-ring. The tasks are not entirely symmetric, we report the results for the first. Finally, DP, DPSYN, and DPD are variants of Dining Philosophers. In DP, philosophers take and release forks one by one, whereas in DPSYN they take and release both at once. In DPD, deadlocks are prevented by passing a dictionary.

In all cases except one (DPD) our algorithm needs clearly fewer events than there are reachable markings; in some families (DAC, DPSYN, RING) there are far fewer events. A comparison of DP and DPSYN is instructive. In DP, neighbours can concurrently pick and drop forks. Intuitively, this leads to fewer cases in which the condition  $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$  for cut-off candidates is satisfied. On the other hand, in DPSYN both forks are picked and dropped synchronously, and so no event in  $\mathcal{A}_i$  is concurrent to any event in the neighbouring components, making the unfolding procedure much more efficient.

## 7 Extensions: Divergences and Weights

### 7.1 Divergences

We extend our algorithm so that the summary also contains information about *divergences*. Intuitively, a divergence is a finite trace of the interface after which the system can “remain silent” forever.

► **Definition 8.** Let  $\mathcal{A}_1, \dots, \mathcal{A}_n$  be LTSs with interface  $\mathcal{A}_i$ . A *divergence* of  $\mathcal{A}_i$  is a finite trace  $\sigma \in Tr(\mathcal{A}_i)$  such that  $\sigma = \tau|_{\Sigma_i}$  for some infinite trace  $\tau \in Tr(\mathbf{A})$ .

A *divergence-summary* is a pair  $(\mathcal{S}_i, D)$ , where  $\mathcal{S}_i$  is a summary and  $D$  is a subset of states of  $\mathcal{S}_i$  such that  $\sigma \in Tr(\mathcal{S}_i)$  is a divergence of  $\mathcal{A}_i$  iff some realization of  $\sigma$  in  $\mathcal{S}_i$  leads to a state of  $D$ .

We define the set of divergent conditions of the output of Algorithm 2, and show that it is a correct choice for the set  $D$ .

► **Definition 9.** Let  $\mathcal{N}$  be the output of Algorithm 2. A condition  $s$  of  $\mathcal{N}_i$  is *divergent* if after termination of the algorithm there is  $e \in coc$  with companion  $e'$  such that  $s$  is concurrent to both  $e$  and  $e'$ . We denote the set of divergent conditions by  $DC$ .

► **Theorem 10.** A finite trace  $\sigma \in Tr(\mathcal{S}_i)$  is a divergence of  $\mathcal{A}_i$  iff there is a divergent condition  $s$  of  $\mathcal{N}_i$  such that some realization of  $\sigma$  leads to  $[s]_{\equiv}$ . Therefore,  $(\mathcal{S}_i, [DC]_{\equiv})$  is a divergence-summary.

The proof of this theorem is given in Appendix A.4

### 7.2 Weights

We now consider weighted systems, e.g parallel compositions of weighted LTS. More formally, a weighted LTS  $\mathcal{A}^w = (\mathcal{A}, c)$  consists of an LTS  $\mathcal{A} = (\Sigma, S, T, \lambda, s^0)$  and a cost function  $c : T \rightarrow \mathbb{R}_+$  associating a weight to each transition. A *weighted trace* of  $\mathcal{A}^w$  is a pair  $(\sigma, w)$  where  $\sigma = a_1 \dots a_k$  is a finite trace of  $\mathcal{A}$  and  $w$  is the minimal weight among the paths realizing  $\sigma$ , i.e:

$$w = \min_{\substack{s_0 \dots s_k \in S^{k+1}, s_0 = s^0, \\ t_i = (s_{i-1}, s_i) \in T, \lambda(t_i) = a_i}} \sum_{j=1}^k c(t_j).$$

We denote by  $Tr(\mathcal{A}^w)$  the set of all the weighted traces of  $\mathcal{A}^w$ . The parallel composition  $\mathcal{A}^w = \mathcal{A}_1^w ||_w \dots ||_w \mathcal{A}_n^w$  of the LTS  $\mathcal{A}_1^w, \dots, \mathcal{A}_n^w$  is such that  $A = \mathcal{A}_1 || \dots || \mathcal{A}_n$  and the weight of a global transition  $\mathbf{t} = (t_1, \dots, t_n)$  is:

$$c(\mathbf{t}) = \sum_{t_i \neq * } c_i(t_i).$$

Similarly a *weighted labelled Petri net* is a tuple  $\mathcal{N}^w = (\mathcal{N}, c)$  where  $\mathcal{N} = (\Sigma, P, T, F, \lambda, M_0)$  is a labelled Petri net and  $c : T \rightarrow \mathbb{R}_+$  associates weights to transitions. A weighted trace in such a net is a pair  $(\sigma, w)$  with  $\sigma$  a finite trace of  $\mathcal{N}$  and  $w$  the minimal weight of an occurrence sequence corresponding to  $\sigma$ , where the weight of an occurrence sequence is the sum of the weights of its transitions. By  $Tr(\mathcal{N}^w)$  we denote the set of all the weighted traces of  $\mathcal{N}^w$ .

The branching processes of  $\mathcal{A}_1^w ||_w \dots ||_w \mathcal{A}_n^w$  are defined as weighted labelled Petri nets in the same way as above. Each event being implicitly labelled by an action (as before) and a cost.

Given a finite set of weighted traces  $W$  we define its restriction to the alphabet  $\Sigma$  as

$$W|_{\Sigma} = \{(\sigma, w) : \exists(\sigma', w') \in W, \sigma = \sigma'_{\Sigma} \wedge w = \min_{\substack{(\sigma', w') \in W \\ \sigma'_{\Sigma} = \sigma}} w'\}.$$

As in the non-weighted case we are interested in solving the following summary problem:

► **Definition 11** (Weighted summary problem). Given weighted LTSs  $\mathcal{A}_1^w, \dots, \mathcal{A}_n^w$  with interface  $\mathcal{A}_i^w$ , compute a weighted LTS  $\mathcal{S}_i^w$  satisfying  $Tr(\mathcal{S}_i^w) = Tr(\mathbf{A}^w)|_{\Sigma_i}$ , where  $\mathbf{A}^w = \mathcal{A}_1^w ||_w \dots ||_w \mathcal{A}_n^w$ .

This section aims at showing that the approach to the summary problem proposed in the unweighted case still works in the weighted one. In other words  $\mathcal{S}_i^w$  can be obtained by computing a finite branching process  $\mathcal{N}^w$  of  $\mathbf{A}^w$  (using Definition 6 of cut-off and cut-off candidates and Algorithm 2) then taking the interface projection  $\mathcal{N}_i^w$  of  $\mathcal{N}^w$  on  $\mathcal{A}_i^w$  and folding it. The notion interface projection needs to be slightly modified to take weights into account. The conditions, events, and arcs of  $\mathcal{N}_i^w$  are defined exactly as above, and the cost of an event  $e$  of  $\mathcal{N}_i^w$  is  $c_i(e) = c([e]) - c([e'])$  if the predecessor  $e'$  of  $e$  in  $\mathcal{N}_i^w$  exists and  $c_i(e) = c([e])$  else, where  $c([e]) = \sum_{e_k \in [e]} c(e_k)$  for  $[e]$  the past of  $e$  in the weighted branching process  $\mathcal{N}^w$ .

► **Theorem 12.** *Let  $\mathbf{A}^w = \mathcal{A}_1^w ||_w \dots ||_w \mathcal{A}_n^w$  with interface  $\mathcal{A}_i^w$ . The instance of Algorithm 2 given by Definition 6 terminates and returns a weighted branching process  $\mathcal{N}^w$  such that the folding  $\mathcal{S}_i^w$  of  $\mathcal{N}_i^w$  is a weighted summary of  $\mathbf{A}^w$ .*

The proof of this theorem is given in Appendix A.5.

## 8 Conclusions

We have presented the first unfolding-based solution to the summarization problem for trace semantics. The final algorithm is simple, but its correctness proof is surprisingly subtle. We have shown that it can be extended (with minor modifications) to handle divergences and weighted systems.

The algorithm can also be extended to other semantics, including information about failures or completed traces; this material is not contained in the paper because, while laborious, it does not require any new conceptual ideas.

We conjecture that the condition  $e' \ll e$  in the definition of cut-off candidate can be replaced by  $e' < e$ , if at the same time the algorithm is required to add events in a suitable order. Similar ideas have proved successful in the past (see e.g. [5, 15]).

---

### References

---

- 1 P. Baldan, A. Bruni, A. Corradini, B. König, C. Rodríguez, and S. Schwoon. Efficient unfolding of contextual Petri nets. *TCS*, 449(1):2–22, 2012.
- 2 J. C. Corbett. Evaluating deadlock detection methods for concurrent software. *IEEE Transactions on Software Engineering*, 22:161–180, 1996.
- 3 J.-M. Couvreur, S. Grivet, and D. Poitrenaud. Designing a LTL model-checker based on unfolding graphs. In *Proceedings of ICATPN*, pages 123–145, 2000.
- 4 J. Esparza and K. Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking*. Springer, 2008.
- 5 J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan’s unfolding algorithm. In *Proceedings of TACAS*, pages 87–106, 1996.
- 6 E. Fabre, A. Benveniste, S. Haar, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. *DEDS*, 15(1):33–84, 2005.
- 7 E. Fabre and L. Jezequel. Distributed optimal planning: an approach by weighted automata calculus. In *Proceedings of CDC*, pages 211–216, 2009.
- 8 B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In *Proceedings of ICALP*, pages 52–63, 2010.
- 9 S. Graf and B. Steffen. Compositional minimization of finite state systems. In *Proceedings of CAV*, pages 186–196, 1990.
- 10 D. Harel, O. Kupferman, and M. Y. Vardi. On the complexity of verifying concurrent transition systems. In *Proceedings of CONCUR*, pages 258–272, 1997.
- 11 S. Hickmott, J. Rintanen, S. Thiébaux, and L. White. Planning via Petri net unfolding. In *Proceedings of IJCAI*, pages 1904–1911, 2007.
- 12 C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- 13 V. Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, Newcastle University, 2003.
- 14 V. Khomenko and M. Koutny. LP deadlock checking using partial-order dependencies. In *Proceedings of CONCUR*, pages 410–425, 2000.
- 15 V. Khomenko, M. Koutny, and W. Vogler. Canonical prefixes of Petri net unfoldings. *Acta Informatica*, 40(2):95–118, 2003.
- 16 V. Khomenko, A. Madalinski, and A. Yakovlev. Resolution of encoding conflicts by signal insertion and concurrency reduction based on STG unfoldings. In *Proceedings of ACSO*, pages 57–68, 2006.
- 17 K. McMillan. A technique of state space search based on unfolding. *FMSD*, 6(1):45–65, 1995.
- 18 M. Mukund and M. Sohoni. Gossiping, asynchronous automata and Zielonka’s theorem. Technical Report TCS-94-2, SPIC Science Foundation, 1994.
- 19 S. Römer. *Theorie und Praxis der Netzentfaltungen als Grundlage für die Verifikation nebenläufiger Systeme*. Phd thesis, TU München, 2000.
- 20 A. Valmari. Compositionality in state space verification methods. In *Proceedings of ICATPN*, pages 29–56, 1996.
- 21 G. Winskel. Events, causality and symmetry. *Computer Journal*, 54:42–57, 2011.
- 22 F. Zaraket, J. Baumgartner, and A. Aziz. Scalable compositional minimization via static analysis. In *Proceedings of ICCAD*, pages 1060–1067, 2005.
- 23 W. Zielonka. Notes on finite asynchronous automata. *RAIRO - Theoretical Informatics and Applications*, 21(2):99–135, 1987.

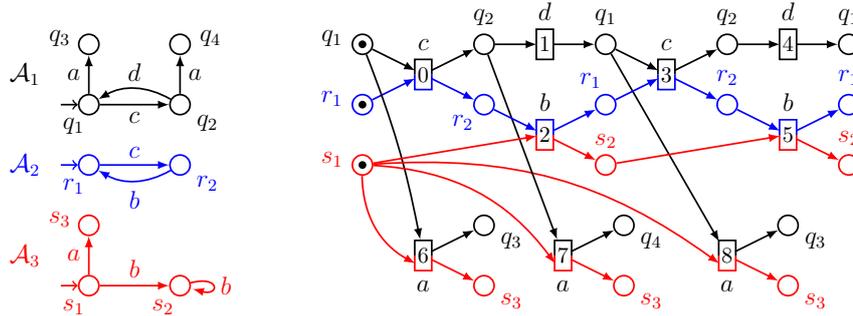
**A Proofs**

► **Definition 13.** Let  $\mathcal{A} = (\Sigma, S, T, \lambda, s^0)$  be a LTS. A sequence of transitions  $\tau = t_1 t_2 t_3 \dots \in T^* \cup T^\omega$  is an *execution* of  $\mathcal{A}$  if there is a sequence  $s_0 s_1 s_2 \dots$  of states such that  $t_k = (s_{k-1}, s_k)$  for every  $k$ . We write  $s_0 \xrightarrow{\tau}$  (or  $s_0 \xrightarrow{\tau} s_n$  when  $\tau$  is finite with  $t_n$  as last transition). An execution is a *history* if  $s_0 = s^0$ . A sequence  $\sigma = a_1 a_2 a_3 \dots \in \Sigma^* \cup \Sigma^\omega$  of actions is a *computation* if there is an execution  $\tau = t_1 t_2 t_3 \dots$  such that  $\lambda(\tau) = \lambda(t_1)\lambda(t_2)\lambda(t_3)\dots = \sigma$ ; if  $s_0 \xrightarrow{\tau}$ , then we also write  $s_0 \xrightarrow{\sigma}$ . (Observe that  $\sigma$  is a trace iff  $\tau$  is a history.) Abusing language, given an execution  $\tau = t_1 t_2 t_3 \dots$ , we denote by  $tr(\tau)$  the computation  $\lambda(t_1)\lambda(t_2)\lambda(t_3)\dots$  (even if it is not necessarily a trace).

**A.1 Standard cut-off condition does not work**

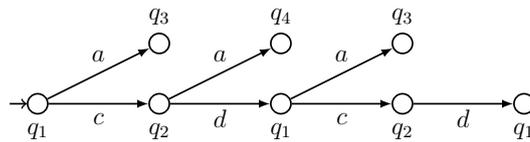
Usually, an event  $e$  is declared a cut-off if the branching process already contains an event  $e'$  with the same global state. If events are added according to an *adequate order* [4], then the prefix generated by the algorithm is guaranteed to contains occurrence sequences leading to all reachable markings.

We show that with this definition of cut-off even we do not always compute a correct summary. We do so by showing an example in which *independently of the order in which Algorithm 1 adds events* the summary is always wrong. Consider the parallel composition of Figure 5 with  $\mathcal{A}_1$  as interface.



■ **Figure 5** Three labeled transition systems (left) and a branching process (right)

Independently of the order in which events are added, the branching process  $\mathcal{N}$  computed by Algorithm 1 is the one shown on the right of Figure 5. The only cut-off event is 5, with companion event 2, for which we have  $\mathbf{St}(5) = \{q_2, r_1, s_2\} = \mathbf{St}(2)$ . The interface projection  $\mathcal{N}_1$  is the transition system in Figure 6.



■ **Figure 6** Projection of the branching process of Figure 5 on  $\mathcal{A}_1$

Since  $\mathcal{N}_1$  does not contain any cut-off, its folding is again  $\mathcal{N}_1$ . Since  $Tr(\mathbf{A})|_{\Sigma_1} \supseteq cdc(dc)^*$ ,  $\mathcal{N}_1$  is not a summary.

## A.2 Proof of Theorem 3

► **Theorem 3.** Let  $\mathbf{A}$  be non-divergent. The instance of Algorithm 1 with cut-off definition 2 terminates with a finite branching process  $\mathcal{N}$ , and the folding  $\mathcal{S}_i$  of  $\mathcal{N}_i$  is a summary of  $\mathbf{A}$ .

**Proof.** Let  $\mathcal{N}$  be the branching process constructed by Algorithm 1. Assume  $\mathcal{N}$  is infinite (i.e., the algorithm does not terminate). Then  $\mathcal{N}$  contains an infinite chain  $e_1 < e_2 \dots$  of causally related events [15]. Since  $\mathbf{A}$  is non-divergent, the infinite configuration  $C = \bigcup_{i=1}^{\infty} [e_i]$  contains infinitely many  $i$ -events. Since the interface  $\mathcal{A}_i$  participates in all of them, they are all causally related, and so  $C$  contains an infinite chain  $e'_1 < e'_2 \dots$  of causally related  $i$ -events. Since  $\mathbf{A}$  has only finitely many global states, the chain contains two  $i$ -events  $e'_j < e'_k$  such that  $\mathbf{St}(e'_j) = \mathbf{St}(e'_k)$ . So  $e'_k$  is a cut-off, in contradiction with the fact that  $e'_{k+1}$  belongs to  $\mathcal{N}$ . So  $\mathcal{N}$  is finite, and so Algorithm 1 terminates.

It remains to prove  $Tr(\mathcal{S}_i) = Tr(\mathbf{A})|_{\Sigma_i}$ . We prove both inclusions separately, but we first need some preliminaries. We extend the mapping  $\mathbf{St}()$  to conditions by defining  $\mathbf{St}(b) = \mathbf{St}(e)$ , where  $e$  is the unique input event of condition  $b$ . Since the states of  $\mathcal{S}_i$  are equivalence classes of conditions of  $\mathcal{N}_i$  and, by definition, if  $b \equiv b'$  then  $\mathbf{St}(b) = \mathbf{St}(b')$ , we can extend  $\mathbf{St}()$  further to equivalence classes by defining  $\mathbf{St}([b]_{\equiv}) = \mathbf{St}(b)$ .

$Tr(\mathcal{S}_i) \subseteq Tr(\mathbf{A})|_{\Sigma_i}$ . Let  $tr^i$  be a (finite or infinite) trace of  $\mathcal{S}_i$ . Then  $[b^0]_{\equiv} \xrightarrow{tr^i}$  in  $\mathcal{S}_i$ , where  $[b^0]_{\equiv}$  is the initial state of  $\mathcal{S}_i$ . By the definition of folding, there exist  $tr_1^i, tr_2^i, tr_3^i, \dots$  (finite sequences of actions) and pairs  $(b'_1, b_1), (b'_2, b_2), (b'_3, b_3), \dots$  of conditions of  $\mathcal{N}_i$  such that (1)  $tr^i = tr_1^i tr_2^i tr_3^i \dots$ ; (2)  $b^0 = b'_1$ ; (3)  $b'_j \xrightarrow{tr_j^i} b_j$  in  $\mathcal{N}_i$  for every  $j$ ; and (4)  $b_{j-1} \equiv b'_j$  for every  $j$ .

By (3) and the definition of projection, we have  $\mathbf{St}(b'_j) \xrightarrow{tr_j} \mathbf{St}(b_j)$  in  $\mathbf{A}$  for some  $tr_j \in \Sigma^*$  such that  $tr_j^i = tr_j|_{\Sigma_i}$ : indeed, if  $e$  and  $e'$  are the input events of  $b_j$  and  $b'_j$ , then  $\mathbf{St}(b_j)$  is reachable from  $\mathbf{St}(b'_{j-1})$  by means of any computation  $tr_j$  corresponding to executing the events of  $[e] \setminus [e']$ , and any such  $tr_j$  satisfies  $tr_j^i = tr_j|_{\Sigma_i}$ . Moreover, by (4) we have  $\mathbf{St}(b_{j-1}) = \mathbf{St}(b'_j)$ . So we get

$$\mathbf{St}(b'_1) \xrightarrow{tr_1} \mathbf{St}(b_1) \xrightarrow{tr_2} \mathbf{St}(b_2) \xrightarrow{tr_3} \dots$$

By (1) and (2) we have  $\mathbf{St}(b^0) \xrightarrow{tr_1 tr_2 tr_3 \dots}$  in  $\mathbf{A}$ , and so  $tr^i = tr|_{\Sigma_i} \in Tr(\mathbf{A})|_{\Sigma_i}$  with  $tr = tr_1 tr_2 tr_3 \dots$ .

$Tr(\mathbf{A})|_{\Sigma_i} \subseteq Tr(\mathcal{S}_i)$ . Let  $tr$  be a finite or infinite trace of  $\mathbf{A}$ . We prove that there exists a trace  $tr^i$  of  $\mathcal{S}_i$  such that  $tr^i = tr|_{\Sigma_i}$ . For that we prove that for every history  $h$  of  $\mathbf{A}$  there exists a history  $h^i$  of  $\mathcal{S}_i$  such that  $tr(h^i) = tr(h)|_{\Sigma_i}$ .

A finite history  $h = \mathbf{t}_1 \dots \mathbf{t}_k$  is *short* if the unique sequence of events of the unfolding  $e_1 \dots e_k$  such that  $\lambda(e_\ell) = \mathbf{t}_\ell$  for every  $\ell \in \{1..k\}$  satisfies the following conditions:  $e_\ell \leq e_k$  for every  $\ell \in \{1..k\}$ , and  $e_k$  is an  $i$ -event. (The name is due to the fact that, loosely speaking,  $h$  is a shortest history in which  $e_k$  occurs.)

We say that a finite or infinite history  $h$  is *succinct* if there are  $h_1, h_2, h_3 \dots$  such that  $h = h_1 h_2 h_3 \dots$ ,  $|tr(h_k)|_{\Sigma_i} = 1$  for every  $k$ , and  $h_1 \dots h_\ell$  is short for every  $\ell$ . We call  $h_1 h_2 h_3 \dots$  the  *$i$ -decomposition* of  $h$ . It is easy to see that for every history there exists a succinct history with the same projection onto  $\mathcal{A}_i$ . ADD AN ARGUMENT HERE? So it suffices to prove the result for succinct histories.

We prove by induction the following stronger result. For every succinct history of  $\mathbf{A}$  with  $i$ -decomposition  $h_1 h_2 h_3 \dots$  there exist  $h_1^i, h_2^i, h_3^i, \dots$  such that for every  $k$ :

- (a)  $H_k^i = h_1^i \dots h_k^i$  is an history of  $\mathcal{S}_i$  such that  $tr(H_k^i) = tr(h_1 \dots h_k)|_{\Sigma_i}$ .

- (b) There exists a configuration  $C_k$  of  $\mathcal{N}$  that contains no cut-offs and such that  $[M(C_k)_i]_{\equiv}$  is the state reached by  $H_k^i$ .

**Base case.** If  $k = 0$ , then  $H_k^i$  is the empty history of  $\mathcal{S}_i$ , take  $C_k = \emptyset$ .

**Inductive step.** Let  $H_{k+1}$  be the prefix of  $h$  with  $i$ -decomposition  $H_{k+1} = h_1 \dots h_k h_{k+1}$  (it is a succinct history of  $\mathbf{A}$ ). Then  $H_k = h_1 \dots h_k$  is succinct with  $i$ -decomposition  $h_1 \dots h_k$ . By induction hypothesis  $H_k^i = h_1^i \dots h_k^i$  and some configuration  $C_k$  satisfy the conditions above.

Let  $o_{k+1} = e_1 \dots e_m$ , where  $m = |h_{k+1}|$ , be the only sequence of events whose labelling is  $h_{k+1}$  and can occur in the order of the sequence from the marking  $M(C_k)$  (this sequence always exists by the properties of  $C_k$ ). Two cases are possible.

1.  $o_{k+1}$  contains no cut-off. In this case  $o_{k+1}$  is a sequence of events from  $\mathcal{N}$  (because  $C_k$  contains no cut-offs). Thus, there exists an execution  $h_{i,k+1}$  of  $\mathcal{S}_i$  from the state  $[M(C_k)_i]_{\equiv}$  to the state  $[M(e_m)_i]_{\equiv}$  such that  $tr(h_{i,k+1}) = tr(h_{k+1})|_{\Sigma_i}$ . So we can take  $h_{k+1}^i = h_{i,k+1}$ . It remains to choose the configuration  $C_{k+1}$ . We take  $C_{k+1}$  as  $C_k \cup \{e_1, \dots, e_m\}$ , which contains no cut-offs because  $C_k$  contains no cut-offs by hypothesis.
2.  $o_{k+1}$  contains some cut-off. Since  $h_k$  is succinct,  $e_m$  is the only  $i$ -event of  $h_{k+1}$ , and the only maximal event of  $\{e_1, \dots, e_m\}$  w.r.t. the causal relation. Since only  $i$ -events can be cut-offs,  $e_m$  is a cut-off, and the only cut-off among the events of  $o_{k+1}$ . So  $o_{k+1}$  is a sequence of events from  $\mathcal{N}$  whose last event is a cut-off. Further, by the maximality of  $e_m$ , the marking reached by  $o_{k+1}$  is  $M(e_m)$ . By the definition of folding,  $\mathcal{S}_i$  has an execution  $h_{i,k+1}$  from the state  $[M(C_k)_i]_{\equiv}$  to the state  $[M(e_m)_i]_{\equiv}$  such that  $tr(h_{i,k+1}) = tr(h_{k+1})|_{\Sigma_i}$ . As above, this allows to take  $h_{k+1}^i = h_{i,k+1}$ .

It remains to choose the configuration  $C_{k+1}$ . We cannot take  $C_{k+1} = C_k \cup \{e_1, \dots, e_m\}$ , because then  $C_{k+1}$  would contain cut-offs. So we proceed differently. We choose  $C_{k+1} = [e'_m]$ , where  $e'_m$  is the companion of  $e_m$ . Since  $e'_m$  is not a cut-off,  $C_{k+1}$  contains no cut-offs. Moreover, since the marking reached by  $o_{k+1}$  is  $M(e_m)$ , we have that  $[M(C_{k+1})_i]_{\equiv}$  is the state reached by  $H_{k+1}^i$ . ◀

### A.3 Proof of Theorem 7

We first prove two properties of the strong causal relation.

► **Lemma 14.** *Every infinite chain  $e_1 < e_2 < e_3 \dots$  of events of a branching process contains a strong causal subchain  $e_{i_1} \ll e_{i_2} \ll e_{i_3} \dots$ .*

**Proof.** Let  $E = \{e_1, e_2, \dots\}$ . Say that a component  $\mathcal{A}_j$  of  $\mathbf{A}$  participates in an event  $e$  if it participates in the transition labelling  $e$ . We partition the (indices of the) components into the set  $S$  of indices  $j$  such that  $\mathcal{A}_j$  participates in finitely many events of  $E$ , and  $\bar{S} = \{1, \dots, n\} \setminus S$ . We say that the LTS  $\mathcal{A}_j$  has *stabilized* at event  $e_k$  in the chain if  $\mathcal{A}_j$  does not participate in any event  $e \geq e_k$ . Let  $e_\alpha$  be any event of  $E$  such that all LTSs of  $S$  have stabilized before  $e_\alpha$ . We claim that there exists  $e_\gamma$  in  $E$  such that  $e_\alpha \ll e_\gamma$ . Since clearly all LTSs of  $S$  have also stabilized before  $e_\gamma$ , a repeated application of the claim produces the desired subsequence. The claim itself is proved in two steps:

- (1) There exists  $e_\beta > e_\alpha$  in  $E$  such that  $M(e_\beta)_k \neq M(e_\alpha)_k$  for every  $k \in \bar{S}$ , (which implies  $M(e_\alpha)_k < e_\beta$  for every  $k \in \bar{S}$ ).

The existence of  $e_\beta$  follows from (1) the fact that all events of  $E$  are causally related, and

- (2) the definition of  $\bar{S}$ , which implies for any  $k \in \bar{S}$  the existence of an infinite subchain  $e_{\ell_1} < e_{\ell_2} < \dots$  such that  $M(e_{\ell_i})_k \neq M(e_{\ell_j})_k$  for every  $i, j$ .

- (2) There exists  $e_\gamma > e_\beta$  in  $E$  such that  $M(e_\gamma)_k > e_\beta$  for every  $k \in \bar{S}$ .  
 Observe that if  $e < M(e_i)_k$  for some  $i$  and some  $k$ , then  $e < M(e_j)_k$  for all  $j > i$  (as  $\forall i, j, \forall k, M(e_i)_k \leq M(e_j)_k$ ). Suppose that  $e_\gamma$  does not exist. Then there exists  $k \in \bar{S}$  such that  $M(e')_k \not> e$  for every  $e' > e$ . As  $k \in \bar{S}$ , there exists, by definition, an infinite subchain  $e < e_{\ell_1} < e_{\ell_2} \dots$  of  $E$  such that  $M(e_{\ell_i})_k \neq M(e_{\ell_j})_k$  for every  $i, j$ . So for any of these  $e_{\ell_i}$  there exists a  $k$ -event  $e'_{\ell_i}$  such that  $e'_{\ell_i} < e_{\ell_i}$  and  $e'_{\ell_i}$  is concurrent with  $e_{\ell_{i-1}}$ . Let  $e''_{\ell_i}$  be an event on a path from  $e'_{\ell_i}$  to  $e_{\ell_i}$  and such that  $b > e$  and  $b' \not> e$  for some  $b, b' \in \bullet e''_{\ell_i}$  (the existence of such an event is ensured by the fact that  $M(e_{\ell_i})_k \not> e$ ). As  $b > e$  we get  $e''_{\ell_i} > e$  and thus  $b'' > e$  for every  $b'' \in e''_{\ell_i} \bullet$ . Hence, by the observation above, the set  $\{k \in \bar{S} : M(e_{\ell_i})_k > e\}$  is strictly greater than the set  $\{k \in \bar{S} : M(e_{\ell_{i-1}})_k > e\}$ . Since  $\mathbf{A}$  is finite, this contradicts the existence of  $k \in \bar{S}$  such that  $M(e')_k \not> e$  for every  $e' > e$  in  $E$ . So the event  $e_\gamma$  exists.

It follows immediately from (1) and (2) that  $e_\alpha \ll e_\gamma$  (because for any  $k, k', M(e_\alpha)_k < e_\beta < M(e_\gamma)_{k'}$ ), and all LTSs of  $S$  have stabilized before  $e_\gamma$ , and so the claim is proved.  $\blacktriangleleft$

► **Lemma 15.** *If  $e' \ll e$  and  $\hat{e}$  is concurrent with both  $e'$  and  $e$ , then  $([e] \setminus [e']) \cap [\hat{e}] = \emptyset$ .*

**Proof.** Assume  $e_1 \in ([e] \setminus [e']) \cap [\hat{e}]$ .

Then  $e_1 \leq e$  and  $e_1 \leq \hat{e}$ . Since  $e$  and  $\hat{e}$  are concurrent, we have  $e \neq e_1 \neq \hat{e}$ . So  $e_1 < \hat{e}$ , and so there is a nonempty path  $e_1 \prec b_1 \prec e_2 \prec b_2 \prec \dots \prec e_k = \hat{e}$ , where  $x \prec y$  denotes  $y \in x \bullet$ . Since  $e$  and  $\hat{e}$  are concurrent, there is a first condition  $b_j$  in the path such that  $b_j$  and  $e$  are concurrent, and we have  $b_j \in M(e)$ . Since  $e_1 \notin [e']$ , we have  $b_j \notin M(e')$ . Since  $e' \ll e$ , we have  $b_j < b$  for every  $b \in M(e') \setminus M(e)$ . In particular, since there is at least one condition  $b'$  such that  $e' \prec b' < e$ , we have  $b_j < b'$ , and so  $e' < b_j$ . But then, since  $b_j$  belongs to the path from  $e_1$  to  $\hat{e}$ , we have  $e' < b_j < \hat{e}$ , contradicting that  $e$  and  $\hat{e}$  are concurrent.  $\blacktriangleleft$

► **Theorem 7.** Let  $\mathbf{A} = \mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$  with interface  $\mathcal{A}_i$ . The instance of Algorithm 2 given by Definition 6 terminates and returns a branching process  $\mathcal{N}$  such that the folding  $\mathcal{S}_i$  of  $\mathcal{N}_i$  is a summary of  $\mathbf{A}$ .

**Proof.** We first prove termination. Assume the algorithm does not terminate, i.e., it constructs an infinite branching process  $\mathcal{N}$ . Then there exists an infinite chain  $e_1 < e_2 < \dots$  of causally related events in  $\mathcal{N}$  [15]. First remark that  $C = \cup_{i=1}^\infty [e_i]$  cannot contain an infinite number of  $i$ -events: if there is infinitely many  $i$ -event in  $C$  one of them must be a cut-off (this is due to the finite number of global states in  $\mathbf{A}$ ) as all the  $i$ -events of  $C$  are causally related there is a contradiction. Hence,  $C$  contains an infinite chain  $w'$  of causally related events such that for any two events  $e$  and  $e'$  of  $w'$  one has  $M(e)_i = M(e')_i$ . From that, the finite number of possible global states in  $\mathbf{A}$  ensures that there exists an infinite subchain  $w''$  of  $w'$  such that for any two events  $e$  and  $e'$  of  $w''$  one has  $\mathbf{St}(e) = \mathbf{St}(e')$ . The finite number of possible global states in  $\mathbf{A}$  also ensures that in  $\mathcal{N}$  there exists only a finite set of non-cut-off  $i$ -events. So, there exists an infinite subchain  $w'''$  of  $w''$  such that for any two events  $e$  and  $e'$  of  $w'''$  one has  $Ico_{\mathcal{N}}(e) = Ico_{\mathcal{N}}(e')$ . Finally, by Lemma 14 there exists two events  $e$  and  $e'$  of  $w'''$  such that  $e' \ll e$ . Then,  $e$  is a cut-off candidate of  $\mathcal{N}$ , which is in contradiction with the infiniteness of  $w'''$  and so with the existence of  $e_1 < e_2 < \dots$ . The termination of Algorithm 2 is thus proved.

Now we prove  $Tr(\mathcal{S}_i) = Tr(\mathbf{A})|_{\Sigma_i}$ . As in the proof of Theorem 3, we extend the mapping  $\mathbf{St}()$  to conditions, and to equivalence classes of conditions of  $\mathcal{N}_i$ .

$Tr(\mathcal{S}_i) \subseteq Tr(\mathbf{A})|_{\Sigma_i}$ . The proof of this part is identical to that of Theorem 3: since the folding  $\mathcal{S}_i$  is completely determined by the cut-offs that are  $i$ -events, and the definition of these cut-offs in Definition 2 and Definition 5 coincide, the same argument applies.

$Tr(\mathbf{A})|_{\Sigma_i} \subseteq Tr(\mathcal{S}_i)$ . The proof has the same structure as the proof of Theorem 3, but with a number of important changes.

Let  $tr$  be a finite trace of  $\mathbf{A}$ . We prove that there exists a trace  $tr^i$  of  $\mathcal{S}_i$  such that  $tr^i = tr|_{\Sigma_i}$ . For that we prove that for every history  $h$  of  $\mathbf{A}$  there exists a history  $h^i$  of  $\mathcal{S}_i$  such that  $tr(h^i) = tr(h)|_{\Sigma_i}$ .

As in Theorem 3, we use the notion of a succinct histories. However, we need to strengthen it even more. Let  $\nu = \mathbf{s}_1 \dots \mathbf{s}_k$  be a sequence of global states of  $\mathbf{A}$ , and let  $H(\nu)$  be the (possible empty) set of succinct histories  $h$  with  $i$ -decomposition  $h_1 \dots h_k$  such that  $\mathbf{s}^0 \xrightarrow{h_1} \mathbf{s}_1 \xrightarrow{h_2} \dots \xrightarrow{h_k} \mathbf{s}_k$ . We say that a history  $h_s \in H(\nu)$  with  $i$ -decomposition  $h_{1s} \dots h_{ks}$  is *strongly succinct* if for every history  $h \in H(\nu)$  with  $i$ -decomposition  $h_1 \dots h_k$  we have  $|h_{js}| \leq |h_j|$  for every  $j \in \{1..k\}$ . Observe that if  $h_1 \dots h_j \dots h_k$  is succinct,  $\mathbf{s}_{j-1} \xrightarrow{h'_j} \mathbf{s}_j$ , and  $|h_j| \leq |h'_j|$ , then  $h_1 \dots h'_j \dots h_k$  is also succinct. Therefore, if  $H(\nu)$  is nonempty then it contains at least one strongly succinct history.

As in Theorem 3, we prove by induction a result implying the one we need. For every strongly succinct history  $h$  of  $\mathbf{A}$  with  $i$ -decomposition  $h = h_1 \dots h_k$ :

- (a) There exists a history  $h^i$  of  $\mathcal{S}_i$  such that  $tr(h^i) = tr(h)|_{\Sigma_i}$ .
- (b) There exists a configuration  $C_h$  of  $\mathcal{N}$  that contains no cut-offs and such that  $[M(C_h)_i] \equiv$  is the state reached by  $h^i$ .
- (c) If  $h$  is nonempty, then there exists an  $i$ -event  $e_h$  such that  $C_h = [e_h]$ .  
(The first two claims are as Theorem 3, while the third one is new.)

**Base case.** If  $h$  is empty, then take  $h^i$  as the empty history of  $\mathcal{S}_i$  and  $C_h = \emptyset$ .

**Inductive step.** The initial part of the inductive step is identical to that of Theorem 3. Let  $h$  be a strongly succinct history of  $\mathbf{A}$  with  $i$ -decomposition  $h = h_1 \dots h_k h_{k+1}$ . Then  $h' = h_1 \dots h_k$  is strongly succinct with  $i$ -decomposition  $h_1 \dots h_k$ . By induction hypothesis there exists a history  $h'^i$ , a configuration  $C_{h'}$ , and, if  $C'_h$  is nonempty, an event  $e_{h'}$  satisfying the conditions above.

Let  $o_{k+1} = e_1 \dots e_m$ , where  $m = |h_{k+1}|$ , be the only sequence of events whose labelling is  $h_{k+1}$  and can occur in the order of the sequence from the marking  $M(C_{h'})$  (this sequence always exists by the properties of  $C_{h'}$ ). Two cases are possible:

1.  $o_{k+1}$  contains no cut-off.

The proof of this case is as in Theorem 3. Part (c) follows because in Theorem 3 we choose  $C_h$  as  $C_{h'} \cup \{e_1, \dots, e_m\}$ , which, since  $e_j \leq e_m$  for every  $j \in \{1..k\}$ , implies  $C_h = [e_m]$ .

2.  $o_{k+1}$  contains some cut-off event.

In Theorem 3 we used the following argument: since  $e_m$  is the only  $i$ -event of  $o_{k+1}$ , and cut-offs must be  $i$ -events,  $e_m$  is a cut-off. This argument is no longer valid, because in Definition 6 non- $i$ -events can also be cut-offs. So we prove that  $e_m$  is a cut-off in a different way.

Let  $e$  be a cut-off of  $o_{k+1}$ , and let  $e'$  be its companion. We prove that, due to the minimality of  $h_{k+1}$  in the definition of strong succinctness, we have  $e = e_m$ .

Assume  $e \neq e_m$ . Since  $e_m$  is the unique  $i$ -event of  $o_{k+1}$ ,  $e$  is not an  $i$ -event. So, by Definition 6, it is an event that became a cut-off candidate and was never freed.

We consider first the case in which  $C'_h$  is the empty configuration. In this case, consider a permutation  $j_1 j_2 j_3$  of  $o_{k+1}$  in which  $j_1$  contains the events of  $[e']$ ,  $j_2$  contains the events of  $[e] \setminus [e']$ , and  $j_3$  contains the rest of the events. Since  $\mathbf{St}(e) = \mathbf{St}(e')$ ,  $h' \lambda(j_1 j_3) = \lambda(j_1 j_3)$  is also a history of  $\mathbf{A}$ . Since  $|j_1 j_3| < |o_{k+1}|$  this contradicts the minimality of  $h_{k+1}$ .

If  $C_{h'}$  is nonempty, then the  $i$ -event  $e_{h'}$  in part (c) of the induction hypothesis exists. We consider the events  $e$  and  $e_{h'}$ . Since  $e_{h'}$  is an  $i$ -event but  $e$  is not, we have  $e \neq e_{h'}$ . Since

there is an occurrence sequence that contains both  $e$  and  $e_{h'}$ , the events are not in conflict. Moreover, since in this occurrence sequence  $e$  occurs after  $e_{h'}$ , we have that  $e$  is not a causal predecessor of  $e_{h'}$  either. So there are two remaining cases, for which we also have to show that they lead to a contradiction:

(b1)  $e_{h'} < e$ . Let  $e'$  be the companion of  $e$ . By the definition of cut-off candidate, we have  $ip(e) = ip(e')$ . Since  $e_{h'}$  is an  $i$ -event and  $e_{h'} < e$ , we have  $e_{h'} < ip(e)$ , and so  $e_{h'} < e' \ll e$ . Consider the permutation  $j_1 j_2 j_3$  of  $o_{k+1}$  in which  $j_1$  contains the events of  $[e'] \setminus [e_{h'}]$ ,  $j_2$  contains the events of  $[e] \setminus [e']$ , and  $j_3$  the rest of the events. Since  $\mathbf{St}(e) = \mathbf{St}(e')$ ,  $h'\lambda(j_1 j_3)$  is also a history of  $\mathbf{A}$ . Since  $|j_1 j_3| < |o_{k+1}|$ , this contradicts the minimality of  $h_{k+1}$ .

(b2)  $e_{h'}$  and  $e$  are concurrent. We handle this case by means of a sequence of claims.

- (i) Let  $e'$  be the companion of  $e$ . The events  $e'$  and  $e_{h'}$  are concurrent.  
Follows from the fact that  $e_{h'}$  is an  $i$ -event and  $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$  by the definition of cut-off candidate.
- (ii)  $([e] \setminus [e']) \cap [e_{h'}] = \emptyset$ .  
Follows from Lemma 15, assigning  $\hat{e} := e_{h'}$ .
- (iii)  $h_{k+1}$  is not minimal, contradicting the hypothesis.  
By (ii), the sets  $[e_{h'}]$  and  $[e] \setminus [e']$  are disjoint. So every event of  $[e] \setminus [e']$  belongs to  $o_{k+1}$ . Consider the permutation  $j_1 j_2 j_3$  of  $o_{k+1}$  in which  $j_1$  contains the events that do not belong to  $[e']$ ,  $j_2$  contains the events of  $[e] \setminus [e']$ , and  $j_3$  the rest. Since  $\mathbf{St}(e) = \mathbf{St}(e')$ ,  $h'\lambda(j_1 j_3)$  is also a history of  $\mathcal{A}$ , and since  $|j_1 j_3| < |o_{k+1}|$  the sequence  $h_{k+1}$  is not minimal.

Since all cases have been excluded, and so we have  $e = e_m$ , i.e., the  $i$ -event  $e_m$  is the unique cut-off of  $o_{k+1}$ . Now we can reason as in Theorem 3. We have that  $o_{k+1}$  is a sequence of events from  $\mathcal{N}$  whose last event is a cut-off, and the marking reached by  $o_{k+1}$  is  $M(e_m)$ . By the definition of folding,  $\mathcal{S}_i$  has an execution  $h_{i,k+1}$  from the state  $[M(C_{h'})_i]_{\equiv}$  to the state  $[M(e_m)_i]_{\equiv}$  such that  $tr(h_{i,k+1}) = tr(h_{k+1})|_{\Sigma_i}$ . This allows to build  $h^i$  as the concatenation of  $h'^i$  and  $h_{i,k+1}$ . We choose  $C_h = [e'_m]$ , where  $e'_m$  is the companion of  $e_m$  and then, obviously  $e_h = e'_m$ . Since  $e'_m$  is not a cut-off,  $C_h$  contains no cut-offs. Moreover, since the marking reached by  $o_{k+1}$  is  $M(e_m)$ , we have that  $[M(C_h)_i]_{\equiv}$  is the state reached by  $h^i$ .  $\blacktriangleleft$

## A.4 Proof of Theorem 10

► **Theorem 10.** Let  $\mathbf{A} = \mathcal{A}_1 \parallel \dots \parallel \mathcal{A}_n$  with interface  $\mathcal{A}_i$ . The instance of Algorithm 2 given by Definition 6 terminates and returns a branching process  $\mathcal{N}$  such that a finite trace  $\sigma$  of the folding  $\mathcal{S}_i$  of  $\mathcal{N}_i$  is a divergence of  $\mathcal{A}_i$  iff there is a divergent condition  $s$  of  $\mathcal{N}_i$  such that some realization of  $\sigma$  leads to  $[s]_{\equiv}$ .

**Proof.** ( $\Rightarrow$ ) Assume that  $\sigma$  is a divergence of  $\mathcal{A}_i$ . By the definition of a divergence, there exists  $\tau \in Tr(\mathbf{A})$  such that  $\tau|_{\Sigma_i} = \sigma$  and  $\tau$  is infinite. So there exists a strongly succinct history  $h$  of  $\mathcal{A}$  such that  $tr(h) = \tau$ . Denote by  $e_i$  the last  $i$ -event of  $h$ . The proof of Theorem 7 guarantees the existence of an  $i$ -event  $e'_i$  in  $\mathcal{N}$  which is not a cut-off and satisfies the following two properties:  $\mathbf{St}(e_i) = \mathbf{St}(e'_i)$ , and there exists a realisation of  $\sigma$  leading to  $[s]_{\equiv}$ , where  $s = M(e_i)_i$ . As  $\tau$  is infinite, the unfolding  $\mathcal{U}$  of  $\mathcal{A}$  contains an infinite occurrence sequence starting at  $M(e_i)$  and containing no  $i$ -event. Since  $\mathbf{St}(e_i) = \mathbf{St}(e'_i)$ , another infinite sequence with the same labelling and without  $i$ -events can occur from  $M(e'_i)$  in  $\mathcal{U}$ . By construction of  $\mathcal{N}$ , and since  $e'_i$  is not a cut-off, a non-empty prefix of this second occurrence sequence appears in  $\mathcal{N}$ , and contains at least one cut-off candidate  $e$ . So  $e$  appears in some occurrence sequence without  $i$ -events starting at  $M(e'_i)$ . It follows that  $e$  is either (1) concurrent with  $e'_i$ , or (2) a successor of  $e'_i$  such that  $ip(e) = M(e'_i)_i$ . Moreover, since  $e$  is not an  $i$ -event, it is

concurrent with  $s = M(e'_i)_i$ . It remains to show that the companion  $e'$  of  $e$  is also concurrent with  $s$ . If (1) holds, i.e., if  $e$  is concurrent with  $e'_i$ , then  $e'$  is concurrent with  $e'_i$  (and so with  $s$ ) as well, because, by the definition of a cut-off candidate, we have  $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$ . If (2) holds, i.e., if  $e > e'_i$ , then we have  $e' > e'_i$  for the same reason as in the case (b1) in the proof of Theorem 7), and so  $e'$  and  $s$  are concurrent.

( $\Leftarrow$ ) Consider a divergent condition  $s$  of  $\mathcal{N}_i$ . By the definition of a divergent condition there exist a cut-off candidate  $e$  with companion  $e'$  such that neither  $e$  nor  $e'$  are i-events, and both  $e$  and  $e'$  are concurrent with  $s$ . Let  $e_i$  be the i-event such that  $M(e_i)_i = s$ . As  $e$  is concurrent with  $s$ , it is either concurrent with  $e_i$ , or a successor of  $e_i$  such that  $ip(e) = M(e_i)_i$ . We consider these two cases separately.

(1)  $e$  is a successor of  $e_i$  such that  $ip(e) = M(e_i)_i$ . Then  $e'$  is a successor of  $e_i$  for the same reason as in case (b1) of Theorem 7. So we have  $[e_i] \subseteq [e'] \subseteq [e]$ . Let  $j_1$  be any occurrence sequence starting from  $M(e_i)$  and containing exactly the events in  $[e'] \setminus [e_i]$  (so  $j_1$  contains no i-events). Let  $j_2$  be any occurrence sequence starting at  $M(e')$  and containing exactly the events in  $[e] \setminus [e']$  (so  $j_2$  contains no i-events either). As  $\mathbf{St}(e) = \mathbf{St}(e')$ , there exists an occurrence sequence  $j_2^1$  in  $\mathcal{U}$  starting at  $M(e)$  and such that  $tr(j_2^1) = tr(j_2)$ ; moreover the last event  $e^1$  of  $j_2^1$  satisfies  $\mathbf{St}(e^1) = \mathbf{St}(e)$ . So we can iteratively construct occurrence sequences  $j_2^k$  for every  $k > 1$ , each of them starting at  $M(e^{k-1})$ , satisfying  $tr(j_2^k) = tr(j_2)$ , and ending with an event  $e^k$  satisfying  $\mathbf{St}(e^k) = \mathbf{St}(e)$ . So the infinite occurrence sequence  $j_1 j_2 j_2^1 j_2^2 \dots$  can occur in  $\mathcal{U}$  from  $M(e_i)$ .

(2)  $e$  is concurrent with  $e_i$ . Then  $e'$  is also concurrent with  $e_i$ , because the definition of a cut-off candidate requires  $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$ . By Lemma 15 we have  $[e_i] \cap ([e] \setminus [e']) = \emptyset$ . Let  $j_1$  be any occurrence sequence starting from  $M(e_i)$  and containing exactly the events in  $[e'] \setminus [e_i]$  (so  $j_1$  contains no i-events).

Given two arbitrary concurrent events  $e_1, e_2$ , let  $M(e_1, e_2)$  be the unique marking reached by any occurrence sequence that fires exactly the events of  $[e_1] \cup [e_2]$ . Let  $j_2$  be any occurrence sequence starting from  $M(e_i, e')$  and containing exactly the events in  $[e'] \setminus [e]$  (so  $j_2$  contains no i-events). As  $\mathbf{St}(e) = \mathbf{St}(e')$  and  $[e_i] \cap ([e] \setminus [e']) = \emptyset$ , there exists an occurrence sequence  $j_2^1$  in  $\mathcal{U}$  starting at  $M(e_i, e)$  and such that  $tr(j_2^1) = tr(j_2)$ ; moreover the last event  $e^1$  of  $j_2^1$  satisfies  $\mathbf{St}(e^1) = \mathbf{St}(e)$ . So for every  $k > 1$  we can iteratively construct sequences  $j_2^k$  starting from  $M(e_i, e^{k-1})$  such that  $tr(j_2^k) = tr(j_2)$  and ending with an event  $e^k$  satisfying  $\mathbf{St}(e^k) = \mathbf{St}(e)$ . It follows that the infinite occurrence sequence  $j_1 j_2 j_2^1 j_2^2 \dots$  can occur in  $\mathcal{U}$  from  $M(e_i)$ .

So in both cases  $\mathbf{A}$  has an infinite execution  $h'$  starting at  $\mathbf{St}(e_i)$  and such that  $tr(h')|_{\Sigma_i}$  is empty. Moreover, if some realization of  $\sigma$  leads to  $[s]_{\Xi} = M(e_i)_i$ , the proof of Theorem 7 guarantees the existence of a history  $h$  of  $\mathbf{A}$  reaching state  $\mathbf{St}(e_i)$  and satisfying  $tr(h)|_{\Sigma_i} = \sigma$ . Taking  $\tau = tr(hh')$  concludes the proof.  $\blacktriangleleft$

## A.5 Proof of Theorem 12

► **Theorem 12.** Let  $\mathbf{A}^w = \mathcal{A}_1^w \parallel_w \dots \parallel_w \mathcal{A}_n^w$  with interface  $\mathcal{A}_i^w$ . The instance of Algorithm 2 given by Definition 6 terminates and returns a weighted branching process  $\mathcal{N}^w$  such that the folding  $\mathcal{S}_i^w$  of  $\mathcal{N}_i^w$  is a weighted summary of  $\mathbf{A}^w$ .

**Proof.** The termination is granted by Theorem 7 as well as the fact that the weighted trace  $(tr, w)$  belongs to  $Tr(\mathcal{S}_i^w)$  if and only if, for some  $w'$ , the weighted trace  $(tr, w')$  belongs to  $Tr(\mathbf{A}^w)|_{\Sigma_i}$ . It remains to show that for any  $tr$  such that  $(tr, w) \in Tr(\mathcal{S}_i^w)$  and  $(tr, w') \in Tr(\mathbf{A}^w)|_{\Sigma_i}$  one has  $w = w'$ . In the following we denote by  $c_i$  the costs functions

of  $\mathcal{S}_i^w$  and  $\mathcal{N}_i^w$ , and by  $c$  the cost function of  $\mathbf{A}^w$ . Similarly we denote by  $\lambda_i$  the labelling function of  $\mathcal{N}_i^w$  and by  $\lambda$  the labelling function of  $\mathbf{A}^w$ .

$w' \leq w$ . This part of the proof is very close to the proof of the first inclusion of Theorem 3. Let  $(tr^i, w)$  be a finite weighted trace of  $\mathcal{S}_i^w$ . Then  $[b^0]_{\equiv} \xrightarrow{tr^i} [b]_{\equiv}$  in  $\mathcal{S}_i^w$  with  $c_i(tr^i) = w$ , where  $[b^0]_{\equiv}$  is the initial state of  $\mathcal{S}_i$ , and  $[b]_{\equiv}$  is some state of  $\mathcal{S}_i$ . By the definition of folding, there exist  $\tau_1^i, \dots, \tau_k^i$  occurrence sequences of  $\mathcal{N}_i$  and  $(b'_1, b_1), \dots, (b'_k, b_k)$  pairs of conditions of  $\mathcal{N}_i$  such that (1)  $tr^i = \lambda_i(\tau_1^i)\lambda_i(\tau_2^i)\dots\lambda_i(\tau_k^i)$ ; (2)  $b^0 = b'_1$  and  $b_k = b$ ; (3)  $b'_j \xrightarrow{\tau_j^i} b_j$  in  $\mathcal{N}_i$  for every  $j = 1, \dots, k$ ; (4)  $b_{j-1} \equiv b'_j$  for every  $j \in \{1..k\}$ ; and (5)  $c_i(\tau_1^i) + \dots + c_i(\tau_k^i) = c_i(tr^i) = w$ .

By (3) and the definition of projection, we have  $\mathbf{St}(b'_j) \xrightarrow{\tau_j} \mathbf{St}(b_j)$  in  $\mathbf{A}$  for some execution  $\tau_j$  such that  $\lambda_i(\tau_j^i) = \lambda(\tau_j)|_{\Sigma_i}$  and  $c(\tau_j) = c_i(\tau_j^i)$ : indeed, if  $e$  and  $e'$  are the input events of  $b_j$  and  $b'_j$ , then  $\mathbf{St}(b_j)$  is reachable from  $\mathbf{St}(b'_{j-1})$  by means of any execution  $\tau_j$  corresponding to executing the events of  $[e] \setminus [e']$ , and any such  $\tau_j$  satisfies  $\lambda_i(\tau_j^i) = \lambda(tr_j)|_{\Sigma_i}$  and  $c(\tau_j) = c(\tau_j^i)$ . Moreover, by (4) we have  $\mathbf{St}(b_{j-1}) = \mathbf{St}(b'_j)$ . So we get

$$\mathbf{St}(b'_1) \xrightarrow{\tau_1} \mathbf{St}(b'_2) \xrightarrow{\tau_2} \dots \xrightarrow{\tau_{k-1}} \mathbf{St}(b'_k) \xrightarrow{\tau_k} \mathbf{St}(b_k)$$

By (1) and (2) we have  $\mathbf{St}(b^0) \xrightarrow{\tau_1 \dots \tau_k} \mathbf{St}(b)$  in  $\mathbf{A}$ , so  $tr^i = tr|_{\Sigma_i} \in Tr(\mathbf{A})|_{\Sigma_i}$  with  $tr = \lambda(\tau_1)\dots\lambda(\tau_k)$ , and by (5) and the definition of a weighted trace  $w' \leq c(tr) \leq c(\tau_1) + \dots + c(\tau_k) = c_i(tr_1^i) + \dots + c_i(tr_k^i) = w$ .

$w \leq w'$ . This part of the proof is almost exactly the same as the proof of the second inclusion of Theorem 7. We describe here the few differences between these two proofs. The main one is the definition of strongly succinct histories: instead of requiring  $|h_{js}| \leq |h_j|$  we require  $c(h_{js}) < c(h_j)$ , or  $c(h_{js}) = c(h_j)$  and  $|h_{js}| \leq |h_j|$ . Then, as we are interested in weights, claim (a) of the induction hypothesis has the supplementary requirement that  $c_i(h^i) = c(h)$ . The base case is then the same, just remarking that the cost of the empty history is 0 in both  $\mathcal{S}_i^w$  and  $\mathbf{A}^w$ . For the inductive step two things have to be done: (1) ensuring that when  $o_{k+1}$  contains a cut-off it is necessarily  $e_m$  and (2) ensuring the new part of claim (a) about weights. For (1) just remark that in all cases  $j_1j_3$  is such that  $c(j_1j_3) \leq c(j_1j_2j_3)$  and  $|j_1j_3| < |j_1j_2j_3|$  so the same arguments as previously can be used with the new definition of a strongly succinct history. For (2) notice that when  $e_m$  is a cut-off i-event, in the unfolding of  $\mathbf{A}^w$  the events that can occur from  $M(e_m)$  and from  $M(e'_m)$  do not only have the same labelling: they in fact correspond to the exact same transitions of  $\mathbf{A}^w$  and so they also have the same weights.

Reusing this proof we have shown that the weighted trace  $(tr, w')$  of  $\mathcal{A}^w$  is such that there exists a history  $h^i$  of  $\mathcal{S}_i^w$  such that  $tr|_{\Sigma_i} = tr(h^i)$  and  $c(h^i) = c(tr) = w'$ . So, by the definition of a weighted trace it comes directly that  $w \leq c(h^i) = w'$ .  $\blacktriangleleft$

## B Algorithmic additions to the unfolding algorithm

As stated in Section 6, we extended the tool MOLE to implement Algorithm 2. In this appendix we detail the work necessary to handle cut-off candidates. Note that the input of our tool is the Petri net representation of a product  $\mathbf{A}$  in which every place is annotated with the component it belongs to.

Inspired by Definition 6, we introduce a *blocking* relation between events: we write  $e' \vdash_{\mathcal{N}} e$  if  $e' \ll e$ ,  $\mathbf{St}(e) = \mathbf{St}(e')$ ,  $ip(e) = ip(e')$ , and  $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$ , in other words  $e$  is a cut-off candidate because of  $e'$ ; let  $\vdash_{\mathcal{N}} e := \{e' \in \mathcal{N} \mid e' \vdash_{\mathcal{N}} e\}$ . Notice that  $\vdash_{\mathcal{N}} e \subseteq [e]$ . Therefore, an over-approximation of this set can be computed when  $e$  is discovered as a possible extension,

by checking all its causal predecessors. When  $\mathcal{N}$  is expanded,  $\vdash_{\mathcal{N}} e$  can only decrease because adding an event may lead to a violation of the condition  $Ico_{\mathcal{N}}(e) \subseteq Ico_{\mathcal{N}}(e')$ .

The blocking relation requires two principal, interacting additions to the unfolding algorithm:

- (i) a traversal of  $[e]$  collecting information about the ‘cut’  $M(e)$ ;
- (ii) computing the concurrency relation between events.

For (i), we modify the way MOLE determines  $\mathbf{St}(e)$ : it performs a linear traversal of  $[e]$ , marking all conditions consumed and produced by the events of  $[e]$ , thus obtaining  $M(e)$ . We extend this linear traversal with Algorithm 3, which computes  $cut = M(e)$ , allowing to directly determine the conditions  $\mathbf{St}(e) = \mathbf{St}(e')$  and  $ip(e) = ip(e')$ . Moreover, every condition  $b$  becomes annotated with a set  $ind(b) := \{j \mid b \leq M(e)_j\}$ . This, together with  $M(e)$  and  $M(e')$ , allows to efficiently determine whether  $e' \ll e$  holds. Notice that if the number of components in  $\mathbf{A}$  is “small”, the operations on  $ind(b)$  can be implemented with bitsets. Thus, the additional overhead of Algorithm 3 with respect to the previous algorithm can be kept small.

---

**Algorithm 3** Traversal of  $[e]$  for efficiently determining  $\vdash_{\mathcal{N}} e$ , where  $i(b)$  denotes the component to which condition  $b$  belongs.

---

```

let  $\mathcal{N}$  be the current branching process and  $e$  its latest extension
set  $worklist := [e]$  and  $cut := \emptyset$ 
for all conditions  $b$ , let  $b$  unmarked and  $ind(b) := \emptyset$ 
while  $worklist \neq \emptyset$  do
  remove a  $<$ -maximal element  $e$  from  $worklist$ 
  add all unmarked conditions  $b \in e^\bullet$  to  $cut$  and set  $ind(b) := \{i(b)\}$ 
   $I := \bigcup_{b \in e^\bullet} ind(b)$ ;
  mark all conditions  $b \in \bullet e$  and set  $ind(b) := I$ 
end while
add all unmarked initial conditions  $b$  to  $cut$  and set  $ind(b) := \{i(b)\}$ 

```

---

Concerning (ii), we are interested in determining the sets  $Ico_{\mathcal{N}}(e)$  for all events  $e$ . We make use of the facts that:

- MOLE already determines, for every condition  $b$ , a set of other conditions  $par(b)$  that are concurrent with  $b$ . When the  $\mathcal{N}$  is extended with event  $e$ , it computes the set  $I := \bigcup_{b \in e^\bullet} par(b)$  and sets  $par(b') = I \cup e^\bullet \setminus \{b'\}$  for every  $b' \in e^\bullet$ .
- Two events  $e, e'$  of  $\mathcal{N}$  are concurrent iff their inputs  $\bullet e$  and  $\bullet e'$  are disjoint and pairwise concurrent. Thus, when  $e$  is added, this relation can be checked by marking the events in  $I$  and checking whether  $I$  includes  $\bullet e'$ . Thus,  $Ico_{\mathcal{N}}(e)$  can be obtained with small overhead w.r.t. the existing implementation.
- At the same time, we can easily determine whether the addition of an event  $e$  should lead to the removal of some event  $e'$  from  $\vdash_{\mathcal{N}} e''$ ; if this causes  $\vdash_{\mathcal{N}} e''$  to become empty,  $e''$  is freed.

## C More complete experimental results

All reported times are on a machine with a 2.8 MHz Intel CPU and 4 GB of memory running Linux. For each example, we report in Table 2 the time it took to compute a summary, the number of events (including all cutoffs), the number of states in the resulting summary  $\mathcal{S}_i$ ,

the size of a minimal deterministic automaton for a summary (Min), and the number of reachable markings.

Test case	Time/s	Events	$ S_i $	Min.	Markings
CYCLIC(6)	0.04	426	5	2	639
CYCLIC(9)	0.17	3347	5	2	7423
CYCLIC(12)	4.04	26652	5	2	74264
CYCLIS(6)	0.05	303	11	5	639
CYCLIS(9)	0.12	2328	11	5	7423
CYCLIS(12)	2.38	18464	11	5	74264
DAC(9)	0.02	86	4	4	1790
DAC(12)	0.03	134	4	4	14334
DAC(15)	0.03	191	4	4	114686
DP(6)	0.06	935	20	4	729
DP(8)	0.22	5121	28	4	6555
DP(10)	2.23	31031	36	4	48897
DPD(4)	0.10	2373	114	6	601
DPD(5)	0.71	23789	332	6	3489
DPD(6)	17.68	245013	903	6	19861
DPSYN(10)	0.02	176	2	2	123
DPSYN(20)	0.07	701	2	2	15127
DPSYN(30)	0.26	1576	2	2	1860498
RING(5)	0.07	511	53	10	1290
RING(7)	0.12	3139	101	10	17000
RING(9)	0.93	16799	165	10	211528

■ **Table 2** More experimental results