Implementation of a complete prefix unfolder for contextual nets

César Rodríguez

Under supervision of Stefan Schwoon

LSV – ENS Cachan

August 2010

General context

In[1] the domain of *verification techniques* for *concurrent systems*, and particularly within the approach of *state space methods*, the *state explosion problem* refers to the computational difficulty of exploring a representation of the system's state space due to its intractably large size [2].

Unfoldings are mathematical structures that can be used to explore the state space of the system while being in some cases exponentially smaller than it. Unfoldings were introduced for Petri nets by Nielsen, Plotkin and Winskel in [7]. While a Petri net unfolding is usually infinite, McMillan observed that a finite part of it can be effectively used for verification purposes [6].

In this work, we develop the first implementation of an unfolding tool (unfolder) for *contextual nets*, i.e., Petri nets extended with *read arcs*, following the abstract approach specified by Baldan et al. in [1].

The problem

The goal of this work was providing the *first implementation of the theory of contextual net unfolding as presented in [1].* For that purpose, we had to first define a concrete algorithm conforming to the abstract specification provided in that reference. In the development of the unfolder, the first objective was *correction of the tool*; secondarily, *performance.*

The main motivation for this work relies on the fact that the last aim of a formal verification method should be that one of being used within some verification tool. Frequently, the first implementation of a theoretical method brings new insights to the theory, that can in turn be used to refine it.

The problem of contextual net unfoldings is not new (see above); the implementation of the contextual unfolder do is. We believe that our contribution is original because we provide new algorithms that concretize the abstract ones in [1], as well as because our tool is, up to our knowledge, the first contextual net unfolder.

[1]This work was supported by Fundación Caja Madrid under the grant *Beca de Postgrado Fundación Caja Madrid.*

The contributions

In the practical arena, our main contribution is a 4700 lines of code *tool* written from scratch in C language, that is able to compute a complete prefix of the full unfolding of a 1-safe contextual net. Additionally, we have filled the gaps left by the abstract specification of the unfolding algorithm of [1], by providing a *concrete algorithm*, as well as *data structures* for representing the important objects utilized by it.

Concerning the theory, we have integrated *adequate* unfolding orders [3] in the framework of contextual unfoldings, proving finiteness of the complete prefix. Remaining extensions of the theory presented in this work have been motivated by our aim to pave the way between *theory and implementation*. We have characterized the notion of conflict between configurations of the unfolding in terms of certain relation that can be efficiently computed. Finally, we have defined the notion of *enriched condition* and provided a characterization to compute efficiently a concurrency relation on enriched conditions, in order to speed up the computation of the unfolding.

Validity of the contributions

It is a great asset of any implementation that one of becoming an *efficient* and *reusable* piece of code. Our tool, even if still in need of some optimization, can accomplish the first target by means of the work on concurrency relations on conditions that we have started. Concerning reusability, a maximum emphasis has been put in the isolation of the different objects (events, conditions, histories, asymmetric conflict relation, etc.) and algorithms operating on them (e.g. conflict between histories, computation of possible extensions), leading to a very *modular* implementation. Additionally, our tool's input format is an extension of that one of the PEP project [8], enabling certain compatibility with other tools.

Theoretical contributions have paved the way between theory and practice of contextual net unfoldings. For instance, we have suggested that the *direct* asymmetric conflict is an interesting notion with regards to the implementation. In the same way, our characterization of the concurrency relation on enriched conditions (§3.5) is an starting point for the optimization of our tool.

Evaluation and perspectives

Performance of our tool is still low compared to other unfolders. For instance, on Petri nets (without read arcs) the Mole unfolder [9] has shown to operate one order of magnitude faster on certain examples. Profiling of our tool has pointed out that the bottleneck is located at the computation of the *possible extensions* to the prefix (§3.1), in particular due to a certain combinatorial blow up in the search of a possible extension. At this moment, our effort is focused on the discovery and implementation of suitable *concurrency relations* that can alleviate such search procedure. In summary, the application of the theory in [1] in this work work has provided insights about where the optimization effort should go.

Our general aim to make of our tool a competitive unfolder, and particularly the research on concurrency relations commented above, will be the first steps to pursue in the Ph.D of the author, starting at September 2010.

1 Introduction

Petri nets are a formalism that has been fruitfully used for the modeling and verification of concurrent systems. They naturally make possible to express notions such as concurrency, causality or conflict between actions or resources [3]. Verification of safety problems for finite state Petri nets can be reduced to reachability analysis, an approach that is feasible in practise whenever we can avoid the state explosion problem. net unfoldings are a partial-order semantics initially introduced for Petri nets by Nielsen, Plotkin and Winskel [7].

The unfolding of a Petri net is another particularly simple, acyclic Petri net so called *occurrence net*, that is behaviourally equivalent to the first. While the unfolding is frequently infinite, McMillan noticed [6] that it is still possible to build a finite prefix of it, containing information enough to be useful for a given application. Such a prefix is called *complete*, in the sense that it represents complete information of the original net according to some criterion, for instance, reachable markings. Finite and complete prefixes can be exponentially smaller than the full reachability graph of the original Petri net, while being still adequate for reachability analysis (in particular *coverability*). Other uses of unfoldings include *planning* [4] or *deadlock detection* [6].

Contextual nets are Petri nets with *read arcs*, i.e., arcs allowing a transition to read a place and not consume it. When interested in reachability analysis, a contextual net can be translated into a Petri net by replacing each read arc by a consume/produce loop. The resulting net is equivalent in terms of its reachability set, and can be unfolded through the same procedure as for Petri nets. However, such unfolding would explicitly enumerate *every* possible interleaving of all the transitions reading from the same place, leading frequently to an explosion in size. This blow up can be avoided if we define the unfolding of a contextual net as another *contextual* net, whose construction procedure is a generalization of the one for plain Petri nets.

Baldan et al. have proposed in [1] such a generalization. In particular, they present an abstract algorithm to compute a complete and finite contextual prefix out of a contextual net. In this work, we refine that abstract algorithm and provide the first implementation of it. Our contextual unfolder consist on a 4700 line of code, C program, written from scratch. The main goals for this implementation were, in this order, correctness and performance.

During the development of our unfolder, some inspiration was taken from the Mole Petri net (without read arcs) unfolder [9]. However, Petri net unfolding and contextual net unfolding are two tasks different enough so that our unfolder could not be an extension of Mole. This is due to the fact that certain fundamental assumptions that can be done in Mole no longer holds in contextual unfolding. For instance, while each transition in a Petri net unfolding implicitly represents *only one* run of the original net, each transition in a contextual unfolding represents a *set* of runs, or *histories*, in the original net. The existence of only one history per event in the unfolding is a central assumption of Mole, that prevented us to reuse most of its algorithms and data structures in our unfolder. Arguments similar to this one lead us to the development of a new tool from scratch.

Implementation of a contextual unfolder have favoured several other contributions. First and foremost, we provide a *concrete algorithm and implementation for the computation of the possible extensions* of the prefix (see §3.1); a *data structure* for the

representation of histories is also provided (§3.3). Concerning the theoretical framework, we generalize (and implement) the notion of *adequate order* [3] from Petri net unfoldings to contextual unfoldings and prove that the complete prefix is still finite (therefore proving termination of our unfolder under this exploration order). We characterize the conflict relation # between configurations (see page 8) in terms of the *direct asymmetric conflict* relation $\uparrow$ (see §3.2), a result of special relevance from the point of view of the implementation. Finally, as a mean to speed up the computation of our unfolder, we define a *concurrency relation* on *enriched conditions* (see §3.5). Subsequently, we characterize this relation in a suitable way that makes possible to use an efficient construction method. Unfortunately, we had no time to implement this approach.

The reminder of the document is mainly divided in three sections. In §2, we introduce contextual nets and their unfoldings. In §3, we present the unfolding algorithm, providing next from subsection §3.1 to §3.5 a detailed description of its auxiliary procedures and data structures. In §4 we summarize how our unfolder has been tested in order to guarantee its correctness. Finally, some conclusions and future research lines are drawn in §5. Due to space constrains, proofs are located in Appendix A.

2 Contextual unfoldings

We introduce in this section the class of contextual nets and its unfoldings. We provide as well, in §2.4, the motivations that led us to the implementation of a new unfolder for contextual nets.

2.1 Contextual nets

A *contextual net* is a tuple $N = \langle P, T, F, C, m_0 \rangle$, where P and T are disjoint sets respectively called *places* and *transitions*, $F \subseteq P \times T \cup T \times P$ is the *flow relation*, $C \subseteq P \times T$ is the *context relation* and $m_0 : P \to \mathbb{N}$ is the *initial marking*. For $x \in P \cup T$, we denote by ${}^\bullet x$ the *preset* of x, defined as $\{y \in P \cup T \mid (y, x) \in F\}$. Similarly, the *postset* of x, written $x^\bullet$, is defined by $\{y \in P \cup T \mid (x, y) \in F\}$. Finally, for a transition $t \in T$ we write $\underline{t}$ to denote the *context* of t, defined as $\{p \in P \mid (p, t) \in C\}$. The same notation is used to denote the context of a place $p \in P$, that is, the set $\{t \in T \mid (p, t) \in C\}$.

Note that contextual nets are an extension of traditional Petri nets. Precisely, Petri nets are the class of contextual nets whose context relation is empty.

A *marking* m is a function $m : P \to \mathbb{N}$ that maps every place of P to a natural number, including 0. If for a marking m and place p, $m(p) = n$, then we say that m marks p with n tokens. If $\{0, \ldots, n\}$ is the range of a marking m, we say that m is n-safe. We will associate, without further comment, every 1-safe marking to a subset of P.

A transition $t \in T$ is said to be *enabled* at a marking m if m marks every place of the preset and context of t with at least one token, i.e., if we have $m(p) > 0$ for $p \in {}^\bullet t \cup \underline{t}$. Any transition enabled at a marking can *fire* (*occur* or *be executed*), leading to a new marking. Specifically, if t is enabled at m, the execution of t at m produces

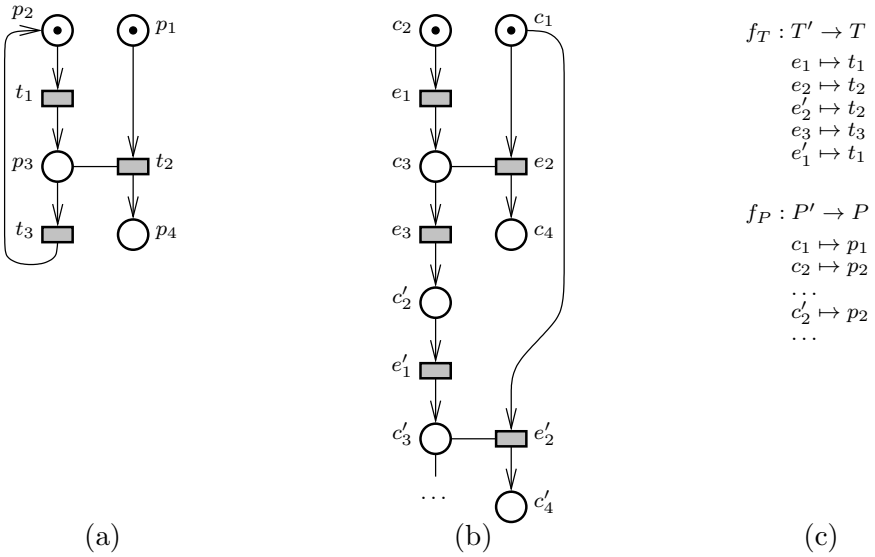


Figure 1: (a) A 1-safe contextual net. (b) A prefix of its full unfolding. (c) The folding morphism.

the new marking

$$m'(p) = \begin{cases} m(p) - 1 & \text{if } p \in {}^\bullet t \setminus t^\bullet \\ m(p) + 1 & \text{if } p \in t^\bullet \setminus {}^\bullet t \\ m(p) & \text{otherwise} \end{cases}$$

A sequence of transitions that can fire one after other starting at the initial marking m_0 is called a *run*. Formally, a finite sequence of transitions $\sigma = t_1 \dots t_n \in T^*$ is a run if there exist markings $m_1, \dots, m_n$ such that t_1 is enabled at the initial marking m_0 and produces m_1 when fired at m_0, and for $2 \leq i \leq n$, transition t_i is enabled at m_{i-1} and produces m_i when fired at m_{i-1}. Marking m_n is said to be the marking reached by σ. Conversely, a marking is said to be *reachable* if there exists some run in the contextual net that reaches it. The set of reachable markings of any contextual net N is denoted by $\mathsf{Markings}(N)$.

By extension, a contextual net N is said to be n-safe if every reachable marking of N is n-safe; it is said *safe* if it is n-safe for some natural number n.

Fig. 1 (a) depicts a contextual net that is 1-safe. Read arcs are represented by means of undirected lines. For t_2, we have that $\{p_1\} = {}^\bullet t_2$, $\{p_3\} = \underline{t_2}$ and $\{p_4\} = t_2^\bullet$. The initial marking is $\{p_1, p_2\}$ and t_1, t_2, t_3 is a run that reaches marking $\{p_2, p_4\}$. Marking $\{p_3, p_4\}$ is reachable because it is the marking reached by the run t_1, t_2.

General assumptions We restrict our interest to finite 1-safe contextual nets. Precisely, when discussing the unfolding of some contextual net $N = \langle P, T, F, C, m_0 \rangle$, we will assume N to be 1-safe and P, T, F and C finite. We make two additional assumptions. First, we assume that no transition in N has an empty preset, i.e., ${}^\bullet t \neq \emptyset$ for all $t \in T$. Second, we assume that for any transition $t \in T$, its preset and context are mutually disjoint, i.e., $\underline{t} \cap {}^\bullet t = \emptyset$.

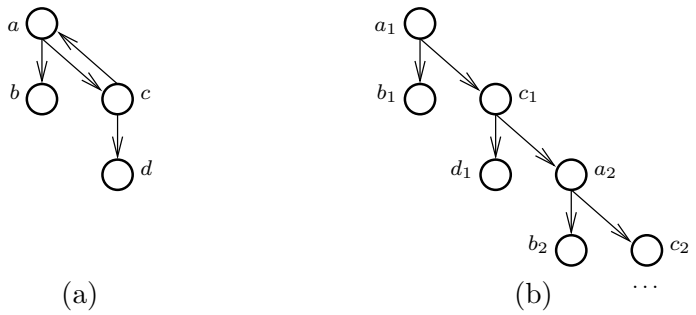


Figure 2: A directed graph (a) together with an arbitrary prefix of its full unfolding (b).

2.2 Unfolding contextual nets

In order to provide an intuitive presentation of the unfolding of a contextual net, let us first introduce the unfolding of a directed graph. Consider a directed graph G and one node v from G. It is well known that one can unfold G into a labelled tree U whose paths from the root node are in one to one correspondence to the paths in G starting from v. A labelling f, associating each node of U to a node of G, implicitly provides this correspondence. Due to the tree-like structure of U, there is a natural well founded precedence order on its nodes (a precedes b if so does in some path from the root).

Construction of the tree is made recursively. We start by appending to U a new node u and updating f with a new binding $f(u) = v$. Then we regard all nodes $v_1, \ldots, v_n$ in G pointed by outgoing edges from v and append to U new nodes $u_1, \ldots, u_n$, as well as the corresponding edges from u. The mapping f is also updated with new bindings $f(u_1) = v_1, \ldots, f(u_n) = v_n$. This construction is next applied recursively to every new node appended to U in the previous step. For this reason, the unfolding operation naturally stops when no new node can be unfolded. The resulting tree U is called the *full* unfolding of G, and it is usually infinite. The algorithm can also be stopped at an arbitrary point, yielding a finite *prefix* of the full unfolding. Fig. 2 illustrates a directed graph (a) and a prefix of its full unfolding (b).

Following the same idea, a contextual net N can also be unfolded into a labelled *contextual occurrence net* $\mathcal{U}_N$, that is, another contextual net whose runs are in in one to one correspondence to the runs of the original net. A labelling, or folding morphism, consist on a pair of functions $\langle f_T, f_P \rangle$ that associates each transition in $\mathcal{U}_N$ to a transition in N (through f_T) and each place of $\mathcal{U}_N$ to a place in N (through f_P). Again, due to the acyclic and rooted structure of the contextual occurrence net, there is a natural precedence (or causal) order on its transitions and places.

Provided N, construction of $\mathcal{U}_N$, as well as f_T and f_P, proceeds incrementally, adding a new transition together with the places in its postset at each step. We start by appending to $\mathcal{U}_N$ a copy of the initial marking of N, and updating f_P to reflect this copy. Then, we consider any reachable marking m' of $\mathcal{U}_N$ and compute its image m though f_P (that is, a marking m in N). If any transition t from N is enabled at m and no copy of t firing from[2] m' is still present in $\mathcal{U}_N$, then a new copy of t

[2]To be precise, we should say 'firing from the places in $\mathcal{U}_N$ whose image through f_P conforms

can be appended, as well as new copies of places in $t^\bullet$. The folding morphisms are subsequently updated to reflect the new transition and places in $\mathcal{U}_N$. Mapping f_T is updated with a binding from the copy of t to t, while mapping f_P is updated to reflect the copy of the postset of t. The same procedure is repeated until no new transition can be appended. The resulting contextual net $\mathcal{U}_N$ is called the *full unfolding*, and it is usually infinite. Nevertheless, the procedure can be stopped at any point, yielding a finite *prefix* of the full unfolding.

The full unfolding $\mathcal{U}_N$, or any prefix of it, is a *contextual occurrence net*, a notion for which we will shortly give a formal definition. Transitions of any contextual occurrence net will, from now, be called *events*, while its places will be called *conditions*. Fig. 1 illustrates a contextual net (a), an arbitrary prefix of its full unfolding (b) and the folding morphism (c).

The full unfolding $\mathcal{U}_N$ is infinite even for trivial examples (see Fig. 1 (b)). For this reason, we are interested in a prefix of $\mathcal{U}_N$ large enough to be *complete*, for some meaning of completeness. In particular, we are interested in computing a prefix whose reachable markings are in one to one correspondence to the reachable markings of N. In order to define this prefix, we need to introduce some technical notions.

For the reminder of this work, we will consider a 1-safe contextual net $N = \langle P, T, F, C, m_0\rangle$ and its full unfolding $\mathcal{U}_N = \langle P', T', F', C', m'_0\rangle$, together with the folding morphism $f_T : T' \to T$ and $f_P : P' \to P$. We let variable t to range transitions in T, variable p for places in P, variable e for events in T' and variable c for conditions in P'.

Definition 1. *The* causality relation *in $\mathcal{U}_N$ is the least transitive relation $<$ on $P' \cup T'$ verifying that, (1) if $c \in {}^\bullet e$, then $c < e$; (2) if $c \in e^\bullet$, then $e < c$; and (3) if $e^\bullet \cap \underline{e'}$, then $e < e'$.*

Intuitively, relation $<$ captures the notion of *what must occur first in* any *run of $\mathcal{U}_N$ that fires or marks some transition or place.* For instance, regarding Fig. 1 (b), we have $c_3 < e_3$ since *any* run firing e_3 will first mark c_3 (notice that all the runs that fire e_3 are prefixed by e_1, e_3 and e_1, e_2, e_3). We also have $c_2 < e_1$ as well as $e_1 < e_2$.

For any condition or event $x \in P' \cup T'$, we denote by $[x]$ the set of *causes* of x, i.e., the set $\{e \in T' \mid e \leq x\}$, where $\leq$ is the reflexive closure of $<$. For instance, $[c_4] = \{e_1, e_2\}$ and $[e_3] = \{e_1, e_3\}$.

Consider now events e_2 and e_3. We cannot say that *any* run that fires e_3 first fires e_2 (as e_1, e_3 is a run). However, we can assure that *if some run fires e_2 and e_3, then it will fire e_2 before e_3* (since once fired e_3, condition c_3 cannot be read). This situation arises due to the existence of read arcs and it is the cause of the existence of several causal histories for certain events. We characterize this relation in the next definition.

Definition 2. *We say that two events $e, e' \in T'$ are in* asymmetric conflict*, and write $e \nearrow e'$, iff either (1) $e < e'$, or (2) $\underline{e} \cap {}^\bullet e' \neq \emptyset$, or (3) $e \neq e' \wedge {}^\bullet e \cap {}^\bullet e' \neq \emptyset$.*

Let us see why the intuition in the previous paragraph holds in the three sub-cases. If $e < e'$, clearly any run that fires both events also fires first e. If $\underline{e} \cap {}^\bullet e' \neq \emptyset$ (the case of e_2 and e_3 in Fig. 1 (b)), then if both e and e' are present in the same run, e must read its context before e' consumes at least one place of it. Finally, observe that

exactly the set ${}^\bullet t$' in N, instead of 'firing from m''.

if $e \neq e'$ and ${}^\bullet e \cap {}^\bullet e' \neq \emptyset$, then e and e' cannot be present in the same run, and the intuition vacuously holds. However, this allows us to capture symmetric conflicts[3] by means of a loop of length two in the asymmetric conflict relation.

For any set of events $X \subseteq T'$, we write $\nearrow_X$ to denote the restriction of $\nearrow$ to X, that is, the relation $\nearrow \cap\, X \times X$.

A *contextual occurrence net* is any 1-safe contextual net N' verifying (1) $|{}^\bullet p| = 1$ for any place p of N'; (2) $<$ is a strict partial order and $[t]$ is finite for any transition t of N'; (3) the initial marking of N' coincides with the set of places p such that ${}^\bullet p = \emptyset$; and (4) $\nearrow_{[t]}$ is acyclic for every transition t of N'. We assume from now that $\mathcal{U}_N$ is a contextual occurrence net, enjoying therefore these four properties.

The next important notion is that one of configuration:

Definition 3. *A finite set of events $C \subseteq T'$ is a* configuration *of $\mathcal{U}_N$ iff (1) $\nearrow_C$ is acyclic and (2) C is causally closed, that is, for all $e' < e$ with $e \in C$, we have $e' \in C$.*

Configurations characterize (concurrent) runs of $\mathcal{U}_N$. A set of events is a configuration iff all its events can be ordered to form a run. Furthermore, any permutation of the events of a configuration that conforms a run is compatible with the asymmetric conflict relation $\nearrow$. In Fig. 1 (b), the set $\{e_1, e_2, e_3\}$ is a configuration, while the set $\{e_2\}$ is not, since it is not causally closed. We let $\mathsf{Conf}(\mathcal{U}_N)$ denote the set of all configurations of $\mathcal{U}_N$.

The *computational order* $\sqsubseteq$ between configurations captures the intuition that a configuration C_1 can evolve to a configuration C_2 if $C_1 \subseteq C_2$ and all events $C_2 \setminus C_1$ can fire after C_1. Formally, we define that $C_1 \sqsubseteq C_2$ holds iff $C_1 \subseteq C_2$ and $\neg(e_2 \nearrow e_1)$ for all $e_1 \in C_1$ and $e_2 \in C_2 \setminus C_1$ holds.

Additionally, two configurations are said to be in *conflict*, written $C_1 \# C_2$, when either $\neg(C_1 \sqsubseteq C_1 \cup C_2)$ or $\neg(C_2 \sqsubseteq C_1 \cup C_2)$. Intuitively, two configurations are in conflict when they cannot evolve to a common configuration. Finally, note that if two configurations are *not* in conflict, then its union is a configuration. The opposite is not necessarily true. Still in Fig. 1 (b), it holds that $\{e_1, e_2\} \sqsubseteq \{e_1, e_2, e_3\}$, while it does not hold that $\{e_1, e_3\} \sqsubseteq \{e_1, e_2, e_3\}$, since $e_2 \nearrow e_3$. Therefore $\{e_1, e_3\} \# \{e_1, e_2, e_3\}$.

As any configuration C is a run of $\mathcal{U}_N$, we can naturally associate to C the marking reached by that run. We call that marking the *cut* of C. In turn, any marking in $\mathcal{U}_N$ corresponds, via f_P, to a marking in N. We call this marking *the marking* of C. Still in Fig. 1 (b), the marking of configuration $\{e_1, e_2\}$ is $\{p_3, p_4\}$, while its cut is $\{c_3, c_4\}$.

Definition 4. *Let C be a configuration of $\mathcal{U}_N$. We define the* cut *of C, written $\mathsf{Cut}(C)$, and the* marking *of C, written $\mathsf{Mark}(C)$ as*

$$\mathsf{Cut}(C) = \left(m \cup \bigcup_{e \in C} e^\bullet \right) \setminus \bigcup_{e \in C} {}^\bullet e \qquad\qquad \mathsf{Mark}(C) = \bigcup_{c \in \mathsf{Cut}(C)} f_P(c)$$

Some configurations are called *histories*. Intuitively, given a configuration C and some event $e \in C$, the history of e in C is the subset of C that contains e as well as all events e' that must fire before e in C, i.e., those for which $e'(\nearrow_C)^* e$ holds.

[3] In a 1-safe Petri net (without read arcs), two transitions are said to be in *symmetric conflict* if they are different and share some place in its respective presets.

Definition 5. *Let C be a configuration of the $\mathcal{U}_N$ and $e \in C$ some event of C. The* history *of e in C, written $C[\![e]\!]$, is the set $\{e' \in C \mid e'(\nearrow_C)^* e\}$. Additionally, we define the set of all histories for any event $e \in T'$, written $\mathsf{Hist}(e)$, as the set $\{C[\![e]\!] \mid C \in \mathsf{Conf}(\mathcal{U}_N) \wedge e \in C\}$.*

We use the notation H_e to denote any history $H_e \in \mathsf{Hist}(e)$ of e. Every history H_e is a configuration. To see this, let C be a configuration and $e \in C$ some event such that $H_e = C[\![e]\!]$. As $H_e \subseteq C$, it is clear that $\nearrow_{H_e}$ is acyclic. To show that H_e is causally closed, assume that $e' \in H_e$ is some event of H_e and that $e'' < e'$. Then we have $e'' \nearrow e'(\nearrow_C)^* e$. As $e'' \in C$, we have $e''(\nearrow_C)^* e$ and therefore $e'' \in H_e$.

In a Petri net unfolding (without read arcs), any event e can only have one history (the so called *local configuration* $[e]$, see [6]). The presence of read arcs gives rise to the existence of multiple histories (or runs) per event, possibly infinite. In Fig. 1 (b), event e_3 have two histories, namely, $\{e_1, e_3\}$ (which coincides with $[e_3]$) and $\{e_1, e_2, e_3\}$.

2.3 A finite and complete prefix of $\mathcal{U}_N$

Due to the fact that $\mathcal{U}_N$ is frequently infinite, we would like to define, and then construct, a finite *prefix* of $\mathcal{U}_N$ that is still *useful*. A finite prefix of $\mathcal{U}_N$ is any contextual occurrence net $\mathcal{P}_N$ that results from stopping at an arbitrary point the abstract unfolding algorithm presented at the beginning of §2.2. All the notions introduced so far in this document for $\mathcal{U}_N$ are likewise applicable to any prefix $\mathcal{P}_N$. Useful, or *complete*, will mean in this work that the prefix is large enough to represent exactly the same reachable markings as N, that is, such that

$$\mathsf{Markings}(N) = \bigcup_{C \in \mathsf{Conf}(\mathcal{P}_N)} \mathsf{Mark}(C)$$

Other definitions are possible, for instance, we could require in addition that all runs of N are represented in $\mathcal{P}_N$ [3]. Under our definition, it is clear that $\mathcal{U}_N$ is complete by construction.

Proposition 6. *The full unfolding $\mathcal{U}_N$ is complete.*

In order to construct a finite and complete prefix of a Petri net (without read arcs) unfolding, McMillan suggested to define certain events of the full unfolding as *cutoff* events, and let the unfolding algorithm to stop either when no new event can be added or when any new event is a cutoff [6]. Then he proved that, for a proper definition of cutoff events, his prefix was finite and complete. Roughly, cutoff events were defined to be those whose corresponding run (history, causal closure or *local configuration*) produces a marking that is reachable by a shorter run (firing less events). His definition of cutoff relies on the fact each event has *only one* history and cannot be adapted without changes to the framework of contextual nets.

The natural generalization of this idea is to consider that cutoffs are, not events, but *enriched events*, that is, pairs $\langle e, H_e \rangle$ where e is an event and $H_e \in \mathsf{Hist}(e)$ is a history of e. The unfolding algorithm has also to be modified to compute, not a prefix of $\mathcal{U}_N$, but an *enriched prefix*, that is, a pair $\langle \mathcal{P}_N, \chi \rangle$ such that $\mathcal{P}_N$ is a prefix of $\mathcal{U}_N$ and $\chi : T' \to 2^{\mathsf{Conf}(\mathcal{U}_N)}$ is a function associating to every event e of $\mathcal{P}_N$ a non empty

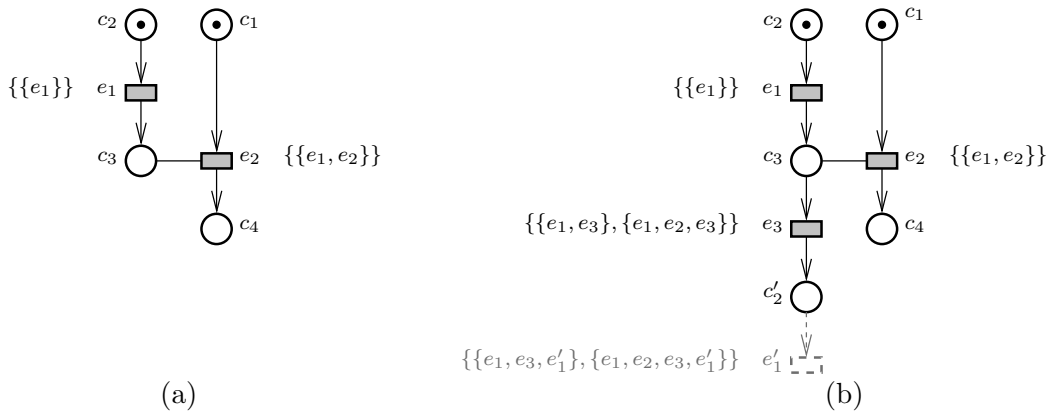


Figure 3: A complete enriched prefix (b) of Fig. 1 (a), together with an incomplete one (a).

set of histories of e (that is, $\emptyset \neq \chi(e) \subseteq \mathsf{Hist}(e)$), additionally verifying[4] that $H \in \chi(e)$ and $e' \in H$ implies $H[\![e']\!] \in \chi(e')$.

Let us set some additional definitions. Let $\mathcal{E}_N = \langle \mathcal{P}_N, \chi \rangle$ be an enriched prefix. First, we set that a configuration C of $\mathcal{E}_N$ is any configuration $C \in \mathsf{Conf}(\mathcal{P}_N)$ that verifies $C[\![e]\!] \in \chi(e)$ for all $e \in C$. The set of configurations of $\mathcal{E}_N$ is denoted by $\mathsf{Conf}(\mathcal{E}_N)$. Second, we consider that, by extension, $\mathcal{U}_N$ *is* the enriched prefix $\langle \mathcal{U}_N, \chi \rangle$ such that $\chi(e) = \mathsf{Hist}(e)$ for $e \in T'$. Finally, we say that $\langle e, H_e \rangle$ is an enriched event *of* $\mathcal{E}_N$, denoted by $\varepsilon \in \mathcal{E}_N$, if e is an event of $\mathcal{P}_N$ and $H_e \in \chi(e)$. We let variable ε to range enriched events. Now we can define a suitable generalization of cutoffs for contextual nets.

Definition 7. *An enriched event $\langle e, H \rangle$ of $\mathcal{U}_N$ is called* cutoff *if either $\mathsf{Mark}(H) = m_0$, the initial marking of N, or there exists another enriched event $\langle e', H' \rangle$ of $\mathcal{U}_N$, so called the* corresponding event*, verifying $\mathsf{Mark}(H) = \mathsf{Mark}(H')$ and $|H| < |H'|$.*

Fig. 3 illustrates two enriched prefixes of the full unfolding of Fig. 1 (a). Histories associated to each event are depicted as sets near the event. The pair $\langle e_3, \{e_1, e_3\} \rangle$ is a enriched event of (b), while it is not an enriched event of (a). Regarding the enriched prefix (b), imagine we remove history $\{e_1, e_2\}$ from $\chi(e_2)$. The resulting prefix would not conform the definition of enriched prefix, since, setting $H = \{e_1, e_2, e_3\}$, we would have that $H \in \chi(e_3)$ and $e_2 \in H$ and not $H[\![e_2]\!] \in \chi(e_2)$, violating the condition stated on χ in the definition of enriched prefix. Finally, note that the enriched prefix (b) is complete, while (a) is not. Indeed, the marking $\{p_2, p_4\}$ is reachable in the original net, reachable in (b) through configuration $\{e_1, e_2, e_3\}$ but unreachable in (a). The two enriched events whose event is e_1' (depicted in gray in (b)) are cutoffs. Precisely, enriched event $\langle e_1', \{e_1, e_3, e_1'\} \rangle$ is a cutoff because its associated marking is the initial

[4]We impose this constrain on χ to conform the definition of *closed occurrence net* of [1]. Intuitively, this constraint forces the enriched prefix to be such that the history of any event e present in the prefix is the union of the histories associated events e' in asymmetric conflict to e. Due to space constrains, we skip providing a detailed motivation of this additional constraint and point the reader to review Definition 12 of [1].

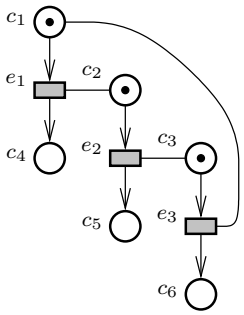


Figure 4: Read arcs can lead to loops of arbitrary length in the $\nearrow$ relation.

marking $\{p_1, p_2\}$. Enriched event $\langle e'_1, \{e_1, e_2, e_3, e'_1\}\rangle$ is a cutoff because its associated marking $\{p_3, p_4\}$ is reachable by means of the smaller history $\{e_1, e_2\}$.

The set of enriched prefixes of the full unfolding $\mathcal{U}_N$ is naturally equipped with with an ordering. Intuitively, enriched prefix $\mathcal{E}_N$ is a *prefix* of $\mathcal{E}'_N$ if the unfolding algorithm can append enriched events to $\mathcal{E}_N$ to reach $\mathcal{E}'_N$. Formally, enriched prefix $\mathcal{E}_N = \langle \mathcal{P}_N, \chi\rangle$ is a prefix of $\mathcal{E}'_N = \langle \mathcal{P}'_N, \chi'\rangle$, written $\mathcal{E}_N \trianglelefteq \mathcal{E}'_N$, iff $\mathcal{P}_N$ is a prefix of $\mathcal{P}'_N$ and for all events e in $\mathcal{P}_N$ we have $\chi(e) \subseteq \chi'(e)$.

At this point, we can properly define the *truncation* of the full unfolding, that is, the greatest enriched prefix free of cutoffs, and state its finiteness and completeness.

Definition 8. *The* truncation *of the full unfolding, denoted $\mathcal{T}_N$, is the greatest enriched prefix of $\mathcal{U}_N$, w.r.t. the $\trianglelefteq$ ordering, which does not contain any enriched event that is a cutoff.*

Theorem 9. *$\mathcal{T}_N$ has a finite number of enriched events and is complete.*

2.4 Challenging aspects of contextual vs Petri net unfoldings

The main goal of this work is to provide an implementation of the unfolding algorithm for contextual nets, presented in [1]. The Mole unfolder [9] served as inspiration for the implementation of our contextual unfolder, but almost[5] no code could be reused from it. In this section, we justify why our implementation is not just an extension of Mole, but a new 4700 lines of code tool written from scratch, by presenting the challenging aspects that arises when unfolding contextual nets instead of Petri nets.

Conflict relation on events. A set of events are in *conflict* iff no configuration (or run) fires all them together. Hence, determining if a set is a configuration amounts to check that there is no subset of events in conflict. In Petri net unfoldings, this can be reduced to a number of pairwise checks of events, relying on the fact that the (symmetric) conflict relation is *binary*. In contextual net unfoldings, this is no longer the case, as we have to check for a loop of *any length* in the (asymmetric) conflict relation. Fig. 4 depicts a contextual net that is equal to its own unfolding, in which we have the loop $e_1 \nearrow e_2 \nearrow e_3 \nearrow e_1$. Events e_1, e_2 and e_3 are hence in conflict, while any

[5]Only the routines to read the description of the contextual net from a file could be reused, as the contextual unfolder uses almost the same input format than Mole.

set of two events is free of conflict (and conforms a configuration). Checking whether a set of events is a configuration is a central task of the unfolding algorithm (see §3.1) and can no longer be made by checking for binary conflicts, as it is done in Mole.

Management of histories. While in the unfolding of a Petri net, each event conceptually represents only one run (or history) of the original net, in the unfolding of a contextual net, events correspond to more than one run in the original net. As the unfolding algorithm used in this work explicitly keeps track of the set of histories associated to each event, we need to implement data structures an algorithms to deal with certain operations performed on histories (for instance duplicate test, or conflict test). There is no explicit notion (data structure or algorithm) of history in Mole that we could reuse or extend.

Concurrency relation on conditions. In the unfolding of a Petri net, each pair of conditions is either in *causal* or *concurrent* or *conflict* relation (see §3.1 at [3]) but never in two of these relations at the same time. In a contextual net unfolding, two conditions can be in (asymmetric) conflict and still be marked by some marking (be concurrent). Conceptually, this prevents us from defining a concurrency relation between conditions. Existence of such relation is successfully exploited in Mole to speed up the computation. For contextual unfoldings, we can still define a concurrency relation on *enriched conditions* (see §3.5), but again, we have to renounce to make an extension of Mole and rather implement such relation from scratch.

Computational order $\sqsubseteq$ and asymmetric conflicts. The computational order $\sqsubseteq$ is remarkably not only set inclusion between configurations, but also requires to check certain conditions regarding the asymmetric conflict relation (see definition in page 8). Our current implementation needs to compute the relation $\sqsubseteq$ between histories each time it extends the unfolding by one enriched event. This requires to explicitly store (a simplified version of) the asymmetric conflict relation between events (see §3.2). No need to store the conflict relation is present in Mole.

3 Computing the unfolding

In the previous section we presented a finite and complete enriched prefix $\mathcal{T}_N$ of the full unfolding $\mathcal{U}_N$ for a 1-safe contextual net N. In this section, we provide an algorithm to compute a possibly larger enriched prefix $\mathcal{F}_N$ that is still finite and complete.

Baldan et al. provide in [1] an abstract algorithm to compute $\mathcal{F}_N$. In the sequel, we detail the data structures and algorithms used to transform this abstract algorithm into a concrete one that can be directly implemented. Let us first introduce the data structure used to store $\mathcal{F}_N$. The unfolding procedure at Algorithm 1 outputs an enriched prefix $\mathcal{F}_N = \langle \mathcal{P}_N, \chi \rangle$, with $\mathcal{P}_N = \langle P', T', F', C', m'_0 \rangle$ and $\chi : T' \to 2^{2^{T'}}$, by encoding the flow and context relations F' and C' in the data structures used for P' and T'. Each event e is represented by means of a tuple $\langle M_p, M_c, t \rangle$, with $M_p, M_c \subseteq P'$ and $t \in T$. Set M_p is the preset of e, set M_c is the context of e and t is the transition of N to which e corresponds. Each condition c is stored by means of a tuple $\langle e, p \rangle$, with $e \in T'$ being an event of $\mathcal{P}_N$ and $p \in P$ the place of N to which c corresponds.

Note that the folding morphisms f_P and f_T are also implicitly encoded in this data structure, and will be assumed to be defined for the conditions and events currently present in the prefix as the algorithm advances. Finally, notice that in order to simplify the algorithm, we make use of a special event $\perp$ whose postset is the initial marking m'_0. Event $\perp$ is *not* mapped through f_T to N and is removed from $\mathcal{F}_N$ at the end of the algorithm.

In Algorithm 1, one realizes that two procedures remain undefined, namely pe_update and is_cutoff. We provide a declarative definition of pe_update in §3.1, while is_cutoff is described here. Procedure is_cutoff(e, H) takes an enriched event as argument and returns true if $\langle e, H\rangle$ is a cutoff, false otherwise. Precisely, is_cutoff(e, H) returns true if it can find an event e' of $\mathcal{F}_N$ and $H' \in \chi(e')$ such that $\mathsf{Mark}(H') = \mathsf{Mark}(H)$ and $|H'| < |H|$, false otherwise[6]. It can be implemented by means of an exhaustive search[7] of such H' in the image of χ.

For the rest of this section, we let $\mathcal{F}_N$ be the enriched prefix computed by Algorithm 1. We let also $\mathcal{E}_N = \langle \mathcal{P}_N, \chi\rangle$, with $\mathcal{P}_N = \langle P', T', F', C', m'_0\rangle$ and $\chi : T' \to 2^{2^{T'}}$, be the enriched prefix that results from *stopping Algorithm 1 at any arbitrary point.* That is, $\mathcal{E}_N$ is any intermediate state of the construction of $\mathcal{F}_N$, and verifies $\mathcal{E}_N \trianglelefteq \mathcal{F}_N$.

3.1 Computation of the possible extensions

Algorithm 1 builds $\mathcal{F}_N$ by starting from an empty enriched prefix $\mathcal{E}_N$ and appending new enriched events to it, one at each iteration of its unique loop. Each enriched event that is a candidate to be appended to $\mathcal{E}_N$ at any point in the construction of $\mathcal{E}_N$ is called a *possible extension.* Intuitively, possible extensions are enriched events of $\mathcal{U}_N$, currently not present in $\mathcal{E}_N$, that are *enabled* at the cut of some configuration of $\mathcal{E}_N$.

Definition 10. *Given an enriched event $\varepsilon = \langle e, H\rangle$ of $\mathcal{U}_N$ that is not an enriched event of $\mathcal{E}_N$, we call ε a* possible extension *of $\mathcal{E}_N$ if $e = \langle M_p, M_c, t\rangle$, and ${}^\bullet t = f_P(M_p)$, and $\underline{t} = f_P(M_c)$, and there is a configuration $C \in \mathsf{Conf}(\mathcal{E}_N)$ verifying $M_p \cup M_c \subseteq \mathsf{Cut}(C)$ and $H = (C \cup \{e\})[\![e]\!]$.*

Additionally, given an enriched event $\varepsilon = \langle e, H\rangle$ of $\mathcal{E}_N$, a possible extension $\varepsilon' = \langle e', H'\rangle$ of $\mathcal{E}_N$ is said to be induced by *ε if $e \nearrow e'$ and $H \subseteq H'$.*

In Algorithm 1, the set E stores the possible extensions of $\mathcal{E}_N$. Every new enriched event ε appended to $\mathcal{E}_N$ inserts at least one new configuration in $\mathcal{E}_N$, possibly rendering E out of date, as new enriched events may now be possible extensions enabled at the cut of such configuration. It is easy to see that any such event is a possible extension induced by ε. Procedure pe_update returns the set of possible extensions to $\mathcal{E}_N$ induced by the enriched event that it takes as argument, and it is used to update E after the addition of every new enriched event. Due to the space constraints, we present here a

[6]Note that Definition 7 declares $\langle e, H\rangle$ as a cutoff if it has a corresponding event $\langle e', H'\rangle$ in $\mathcal{U}_N$, not necessarily in $\mathcal{F}_N$. Our approach works because we use a precise order when appending new enriched events to $\mathcal{F}_N$. In particular, this order assures that when we append an enriched event $\langle e, H\rangle$, all enriched events $\langle e', H'\rangle$ of $\mathcal{U}_N$ with $|H'| < |H|$ have already been appended to $\mathcal{F}_N$. For this reason, if $\langle e, H\rangle$ has a corresponding event in $\mathcal{U}_N$, then that event is already present in $\mathcal{F}_N$.

[7]However, we didn't implement it like that. A more elaborated version of it can be as follows: we store a hash table mapping $\mathsf{Mark}(H_e)$ to $\varepsilon = \langle e, H_e\rangle$ for every ε of $\mathcal{F}_N$. When is_cutoff(e, H) is called, we access the hash table with $\mathsf{Mark}(H)$. If no entry can be found, we return false. Otherwise, if $\langle e', H'\rangle$ is found, we return $H' < H$.

Algorithm 1 Unfolding procedure, see the text.

Require: A 1-safe contextual net $N = \langle P, T, F, C, m_0 \rangle$.

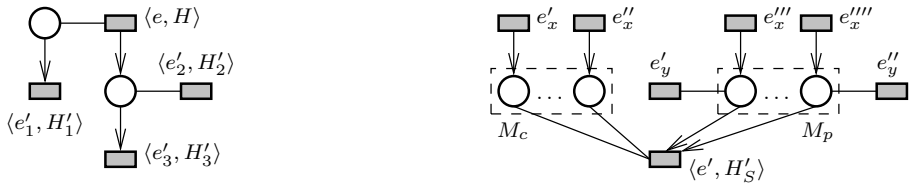
Ensure: A finite and complete enriched prefix $\mathcal{E}_N = \langle \mathcal{P}_N, \chi \rangle$ of the full unfolding of N, with $\mathcal{P}_N = \langle P', T', F', C', m'_0 \rangle$ and $\chi : T' \to 2^{2^{T'}}$.

$T' = \{\bot\}$
$\chi(\bot) = \{\{\bot\}\}$
$m'_0 = \emptyset$
$m'_0 = m'_0 \cup \{\langle \bot, p \rangle\}$ for each $p \in m_0$
$P' = m'_0$
$E = \mathsf{pe\_update}(\bot, \{\bot\})$
while $E \neq \emptyset$ **do**
  Remove from E some $\langle e, H \rangle$ minimal w.r.t $|H|$, and assume that e is $\langle M_p, M_c, t \rangle$
  if not $\mathsf{is\_cutoff}(e, H)$ **then**
    $T' = T' \cup \{e\}$
    $\chi(e) = \chi(e) \cup \{H\}$
    $P' = P' \cup \{\langle e, p \rangle\}$ for each $p \in t^\bullet$
    $E = E \cup \mathsf{pe\_update}(e, H)$
  end if
end while
Remove $\bot$ from T' and remove the binding $\chi(\bot) = \{\{\bot\}\}$.

declarative definition of pe_update. A pseudocode version of the procedure is presented in Appendix B.

In order to compute the possible extensions induced by a given enriched event $\langle e, H \rangle$, we consider enriched events $\langle e', H' \rangle$ with $e \nearrow e'$ (see Fig. 5 (a)). For such e', we compute a history H' as the union of $\{e'\}$ and a family of histories already present in $\mathcal{E}_N$. That family, denoted by $\mathbf{F}_S$, will be a set of histories whose union defines a configuration such that the cut associated to that configuration marks the preset and context of e'.

More precisely, we enumerate all the tuples $e' = \langle M_p, M_c, t \rangle$ verifying that $f_P(M_p) = {}^\bullet t$, and $f_P(M_c) = \underline{t}$, and either $\underline{e} \cap M_p \neq \emptyset$ or $e^\bullet \cap (M_p \cup M_c) \neq \emptyset$. For each e', we compute the sets of events $X = {}^\bullet(M_p \cup M_c)$ and $Y = \underline{M_p}$ (see Fig. 5 (b)). Now, we enumerate all subsets $S \subseteq Y$, and for each one we compute the set $\mathbf{F}_S$, defined as the



(a) (b)

Figure 5: (a) Possible extensions induced by $\langle e, H \rangle$. (b) Computation of new histories for e'.

only family of histories of the events $X \cup S \cup \{e\}$ that verifies

$$\{H\} \subseteq \mathbf{F}_S \subseteq \bigcup_{e'' \in X \cup S \cup \{e\}} \chi(e'') \qquad \text{and} \qquad |\mathbf{F}_S \cap \chi(e'')| = 1 \text{ for all } e'' \in X \cup S \cup \{e\}$$

In other words, $\mathbf{F}_S$ is a set of histories containing history H, one history of *each* $e_x \in X$ and one history of each $e_y \in S$ (or, in other words, one history of *some* $e_y \in Y$). Finally, for each $\mathbf{F}_S$, we generate the enriched event $\langle e', H'_S \rangle$, with

$$H'_S = \{e'\} \cup \bigcup_{H'' \in \mathbf{F}_S} H'' \qquad \text{if} \qquad \neg(H_1 \# H_2) \text{ for all } H_1, H_2 \in \mathbf{F}_S \quad \text{and} \quad \mathsf{Conc}(M_p \cup M_c)$$

The fist condition, $\neg(H_1 \# H_2)$ for all H_1, H_2, assures that $H'_S \setminus \{e'\}$ is a configuration. The second condition, $\mathsf{Conc}(M_p \cup M_c)$, verifies that e' is enabled at $\mathsf{Cut}(H'_S \setminus \{e'\})$, and can be easily computed by checking that no history $H'' \in \mathbf{F}_S$ consumes[8] any condition of $M_p \cup M_c$.

The abstract presentation of [1] subtly omitted to specify explicitly the second condition. Led by this specification, our first version of the contextual unfolder was incorrect. Only exhaustive testing discovered the bug in the pe_update algorithm and led to a modification of the conditions being computed by that procedure, resulting in the addition of the test $\mathsf{Conc}(M_p \cup M_c)$.

3.2 Direct asymmetric conflict

According to Definition 2, any two events e, e' of $\mathcal{P}_N$ in causal relation, $e < e'$, are also in asymmetric conflict relation, $e \nearrow e'$. Transitivity of $<$ would lead to some difficulties if we were interested in storing the $\nearrow$ relation. Furthermore, our current implementation of pe_update uses the relation $\nearrow$ to compute the relation $\#$ between histories. We show in the sequel how we can define, store and use a simplified version of the asymmetric conflict relation that provides, for our purposes, as much information as $\nearrow$.

Definition 11. *Given events e, e' of $\mathcal{U}_N$, we say that e is in* direct asymmetric conflict *to e', and write $e \uparrow e'$, iff either $\underline{e} \cap {}^\bullet e' \neq \emptyset$, or $e \neq e' \wedge {}^\bullet e \cap {}^\bullet e' \neq \emptyset$, or $e^\bullet \cap {}^\bullet e' \neq \emptyset$ or $e^\bullet \cap \underline{e'} \neq \emptyset$*

Fig. 6 (a) illustrates all cases of Definition 11: for any depicted e', we have $e \uparrow e'$. Observe also that, regarding Fig. 5 (a), we have $e \uparrow e'$ for all depicted e'. Another characterization of $\uparrow$ is to say that $e \uparrow e'$ iff $e \nearrow e'$ and $e < e' \implies e^\bullet \cap ({}^\bullet e' \cup \underline{e'}) \neq \emptyset$. This means that relation $\uparrow$ is a subset of $\nearrow$ that excludes the full relation $<$. This is interesting from the point of view of the implementation because computing and storing $\uparrow$ is easier than $\nearrow$, while $\uparrow$ is still useful for our purposes, as we see now. Let us define $C_1 \sqsubseteq_\uparrow C_2$ as $C_1 \subseteq C_2$ and $\neg(e_2 \uparrow e_1)$ for all $e_1 \in C_1$ and all $e_2 \in C_2 \setminus C_1$. Relation $\sqsubseteq_\uparrow$ uses $\uparrow$ instead of $\nearrow$ (see definition of $\sqsubseteq$ at page 8), but it is equivalent to $\sqsubseteq$:

Proposition 12. *For any pair of configurations $C_1, C_2 \in \mathsf{Conf}(\mathcal{U}_N)$, we have $C_1 \sqsubseteq C_2$ iff $C_1 \sqsubseteq_\uparrow C_2$.*

[8] We can say that a configuration (in particular, a history) *consumes* a condition if any run associated to the configuration so does.

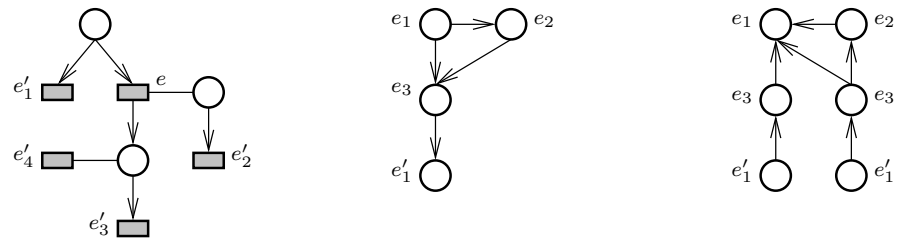


Figure 6: (a) All possible cases of $e \uparrow e'$ in Definition 11. (b) The graph $\mathcal{A}_N$ and (c) the graph $\mathcal{H}_N$ for the (enriched) prefix of Fig. 3 (b).

We store the relation $\uparrow$ on events of $\mathcal{P}_N$ by means of a directed graph $\mathcal{A}_N$, whose set of nodes coincides with the set of events of $\mathcal{P}_N$ and whose set of edges contains a pair (e, e') iff $e \uparrow e'$ (see Fig. 6 (b) for an example). Whenever a new event e is appended to $\mathcal{P}_N$, we use Definition 11 to update $\mathcal{A}_N$. This can be done easily by considering the events reading or consuming ${}^\bullet e$ and $\underline{e}$.

3.3 History graph

Algorithm 1 needs to deal with enriched events $\langle e, H\rangle$ of $\mathcal{E}_N$. We now describe a data structure to store the history H of such enriched events, as well as the mapping χ.

A naive implementation could represent every history by means of the list of events contained in it. Naturally, this would lead to a bottleneck as the prefix $\mathcal{E}_N$ *grows* and histories are larger. We take advantage of the fact that each history H is the union of $\{e\}$ with a set of histories $H_{e'}$ for events $e' \uparrow e$, as explained in §3.1. We represent H by means of a node in the so called *history graph*.

Definition 13. *For a given enriched prefix $\mathcal{E}_N$, we define the* history graph $\mathcal{H}_N = (V, \rightarrow)$ *as the directed graph whose set of nodes V coincides with the set $\{\langle e, H\rangle \mid e \in T' \wedge H \in \chi(e)\}$ of enriched events of $\mathcal{E}_N$, and whose edges are pairs $\langle e, H\rangle \rightarrow \langle e', H'\rangle$ iff $e' \in H$, and $e' \uparrow e$ and $H' = H[\![e']\!]$.*

Graph $\mathcal{H}_N$ stores one node for every enriched event of $\mathcal{E}_N$. The edge relation $\rightarrow$ is closely related to the way in which possible extensions are computed. Regarding again Fig. 5 (b), each (new) history H'_S is the union of $\{e'\}$ with one history $H_{e_x} \in \chi(e_x)$ for each $e_x \in X$ and one history $H_{e_y} \in \chi(e_y)$ for each $e_y \in S$. Grabbing the same structure, $\mathcal{H}_N$ keeps one edge $\langle e', H'_S\rangle \rightarrow \langle e_x, H_{e_x}\rangle$ for each $e_x \in X$ and one edge $\langle e', H'_S\rangle \rightarrow \langle e_y, H_{e_y}\rangle$ for each $e_y \in S$. This gives rise to the next lemma.

Lemma 14. *Given $\mathcal{H}_N = (V, \rightarrow)$ and $\varepsilon = \langle e, H\rangle \in V$, we have $H = \{e' \in T' \mid \varepsilon \rightarrow^* \langle e', H'\rangle\}$.*

Lemma 14 basically says that, given any node ε of $\mathcal{H}_N$, we can build the history that it represents by computing the set of events e' such that $\varepsilon \rightarrow^* \langle e', H'\rangle$. Let $r\colon V \rightarrow 2^{T'}$ be the function that maps each node ε of $\mathcal{H}_N$ to the set $r(\varepsilon) = \{e' \in T' \mid \varepsilon \rightarrow^* \langle e', H'\rangle\}$. Function r can be be easily implemented by means of a search procedure on $\mathcal{H}_N$.

Our unfolder implements the mapping χ by means of the graph $\mathcal{H}_N$ and a list l_e of nodes of $\mathcal{H}_N$ associated to each event $e \in T'$. In order to enumerate $\chi(e)$, it suffices to

enumerate l_e. In order to enumerate the events in the history of some node ε of $\mathcal{H}_N$, we return the set $r(\varepsilon)$.

3.4 Cutoff criterion and adequate orders

The cutoff criterion (Definition 7) is the key factor that makes the truncation $\mathcal{T}_N$ a finite and complete prefix. Other cutoff criteria are possible, still leading to a finite and complete prefix. Esparza et al. proposed in [3] the notion of *adequate order* on configurations of $\mathcal{U}_N$, that is, a characterization of the orders for which we can reformulate the definition of cutoff in such a way that the proof of finiteness and completeness of $\mathcal{T}_N$ still works. We provide now a generalization for contextual nets:

Definition 15. *A partial order $\prec$ on $\mathsf{Conf}(\mathcal{U}_N)$ is called* adequate *iff (1) $\prec$ is well founded, and (2) $C_1 \sqsubset C_2$ implies $C_1 \prec C_2$, and (3) $\prec$ is preserved by finite extensions, that is, if $C_1 \prec C_2$, and $\mathsf{Mark}(C_1) = \mathsf{Mark}(C_2)$, and $C_1 \sqsubset C_1 \cup E$ for some extension E of C_1, and $C_2 \sqsubset C_2 \cup E'$ for some extension E' isomorphic to E, then $C_1 \cup E \prec C_2 \cup E'$.*

In [3], adequate orders are defined for Petri nets. Every adequate order as defined in [3] is an adequate order as defined in Definition 15, which intuitively means that our definition is more general. Briefly, this holds because $C_1 \sqsubset C_2$ implies $C_1 \subset C_2$.

We can redefine now the notion of cutoff according to any order on $\mathsf{Conf}(\mathcal{U}_N)$ that is adequate. We call *adequate cutoff* to any enriched event $\langle e, H\rangle$ of $\mathcal{U}_N$ if either $\mathsf{Mark}(H) = m_0$, the initial marking of N, or there exists another enriched event $\langle e', H'\rangle$ of $\mathcal{U}_N$ verifying $\mathsf{Mark}(H) = \mathsf{Mark}(H')$ and $H \prec H'$ for some adequate order $\prec$. In the same way, we can redefine the truncation of the full unfolding. The *adequate truncation*, denoted by $\widehat{\mathcal{T}}_N$, is the greatest enriched prefix of $\mathcal{U}_N$, w.r.t. the $\trianglelefteq$ ordering, free of adequate cutoffs. Under this notion of cutoff, we can still prove the finiteness and completeness of $\widehat{\mathcal{T}}_N$:

Theorem 16. *$\widehat{\mathcal{T}}_N$ has a finite number of enriched events and is complete.*

The so called McMillan order ($C_1 \prec C_2$ iff $|C_1| < |C_2|$) is adequate. This is the order used in [1] as well as on Algorithm 1 and the definition of is_cutoff. For our unfolder, we also implemented the order $\prec_F$ of [3], which, due to space constraints we cannot present here.

3.5 Optimizations

Experimental verification has shown that the bottleneck in Algorithm 1 is located in the computation of the possible extensions to $\mathcal{E}_N$, in particular in the computation of the # relation. Roughly speaking, code profiling has pointed out that our tool spends in average 85% of the time computing the relation #. We propose now an optimization of procedure pe_update that could speed up the unfolding algorithm.

In a Petri net unfolding, it is possible to define a *binary* concurrency relation R on the unfolding conditions. Then, it is possible to determine if a number of conditions are concurrent by checking whether *every* pair of conditions is in R. Such approach is successfully exploited by Mole to speed up the computation of the unfolding. Unfortunately, this idea cannot be used without modifications in the framework of contextual

unfoldings. We can nevertheless define a concurrency relation, not on conditions, but on enriched conditions.

An *enriched condition* of $\mathcal{E}_N$ is a pair $\rho = \langle c, H\rangle$ such that either $H = \emptyset$ and $c \in m'_0$, the initial marking of $\mathcal{E}_N$, or there is an enriched event $\langle e, H\rangle$ of $\mathcal{E}_N$ verifying $c \in e^\bullet \cup \underline{e}$. We denote by $\rho \in \mathcal{E}_N$ the fact that ρ is an enriched condition of $\mathcal{E}_N$. We need some notation to work with enriched conditions and events. We define the *preset* ${}^\bullet\rho$ of an enriched condition $\rho = \langle c, H\rangle \in \mathcal{E}_N$ as the set $\{\langle e, H'\rangle \in \mathcal{E}_N \mid H' \in \chi(e) \wedge H' = H\}$. Additionally, we define the *preset* ${}^\bullet\varepsilon$ and *context* $\underline{\varepsilon}$ for an enriched event $\varepsilon = \langle e, H\rangle \in \mathcal{E}_N$ as, respectively, the sets $\{\langle c, H'\rangle \in \mathcal{E}_N \mid c \in {}^\bullet e \wedge \exists e' \in H$ maximal w.r.t. $\nearrow_H \wedge H' = H[\![e']\!]\}$ and $\{\langle c, H'\rangle \in \mathcal{E}_N \mid c \in \underline{e} \wedge \exists e' \in H$ maximal w.r.t. $\nearrow_H \wedge H' = H[\![e']\!]\}$.

We define now the binary *concurrency relation* on enriched conditions $\rho = \langle c, H\rangle$, $\rho' = \langle c', H'\rangle$ of $\mathcal{E}_N$ as

$$\rho \parallel \rho' \stackrel{def}{\iff} \neg(H \# H') \wedge \{c, c'\} \subseteq \mathsf{Cut}(H \cup H')$$

That is, ρ is concurrent to ρ' if its histories are not in conflict, and H does not consume c' and H' does not consume c. Relation $\parallel$ enjoy the property that any set of conditions $\{c_1, \ldots, c_n\}$ is concurrent iff there exist enriched conditions $\langle c_1, H_1\rangle, \ldots, \langle c_n, H_n\rangle$ verifying $\langle c_i, H_i\rangle \parallel \langle c_j, H_j\rangle$ for $1 \leq i < j \leq n$. We believe that is remarkably interesting from the point of view of the implementation, since we conjecture that can use $\parallel$ to speed up the computation of pe_update in the same way as it is done in Mole.

In order to use $\parallel$ for the computation of pe_update, one alternative is to store $\parallel$ for *every* pair of enriched conditions currently present in $\mathcal{E}_N$ and to update the relation whenever new enriched conditions ρ are appended to $\mathcal{E}_N$. Furthermore, it is possible to compute the update of $\parallel$ by computing $\rho \parallel \rho'$ for each ρ newly appended to $\mathcal{E}_N$ and every ρ' already present in $\mathcal{E}_N$ *without computing* $H \# H'$. This can be done by means of the next equivalence.

Theorem 17. *Let $\rho = \langle H, c\rangle$, $\rho' = \langle H', c'\rangle$ be two different enriched conditions of $\mathcal{E}_N$ such that $H \in \chi(e)$, $H' \in \chi(e')$ and $H' \prec H$ for an adequate order $\prec$. We have the equivalence*

$$\rho \parallel \rho' \iff \bigwedge_{\rho_i \in {}^\bullet{}^\bullet\rho} \rho_i \parallel \rho' \ \wedge \bigwedge_{\sigma_j \in {}^\bullet\underline{\rho}} \sigma_j \parallel \rho' \ \wedge \ (\rho' \notin {}^\bullet{}^\bullet\rho) \ \wedge \ \neg\exists e'' \in H' \setminus H,\ \underline{e''} \cap {}^\bullet e \neq \emptyset$$

To understand the practical utility of this equivalence, assume that $\langle e, H\rangle$ is the last possible extensions appended to $\mathcal{E}_N$. This triggered the addition of enriched conditions $\rho = \langle c, H\rangle$ with $c \in e^\bullet \cup \underline{e}$. At this point, we use Theorem 17 to compute and subsequently store $\rho \parallel \rho'$ for every ρ' already present in $\mathcal{E}_N$. We compute $\rho \parallel \rho'$ by means of a boolean conjunction that regards the previously computed results of $\rho_i \parallel \rho'$ and $\sigma_j \parallel \rho'$ for certain enriched conditions ρ_i and σ_j, as well as other logical conditions, that are also easy to compute, considering the internal data structures of our unfolder.

The ability to use the relation $\parallel$ to compute sets of concurrent conditions as well as the fact that it can be computed incrementally as $\mathcal{E}_N$ grows make from $\parallel$ an interesting optimization that has not been, for the time being, implemented into our unfolder due to time constraints.

4 Implementation testing

As stated in the introduction, our first priority while implementing the unfolding algorithm was correction. The second, performance. We present here some of the tests that have been performed to the tool.

One of our problems was to know if the unfolding prefix generated after the computation was complete or not. When testing under toy examples, this can be check by hand, but this is no longer the case when the examples are big. We addressed this problem in two different ways.

We made profit of the existence of the Petri net unfolder Mole. Mole is able to compute a compete unfolding prefix for a Petri net, without read arcs. We developed a small tool to check whether two given Petri nets are isomorphic, and compared the resulting prefixes when unfolding the same Petri net with both unfolders. We can assure that for all the (trivial and non trivial) examples provided by the PEP Project [8], the output of our unfolder is a prefix isomorphic to the prefix generated by Mole.

Of course, Mole cannot unfold contextual nets. In order to check our unfolder on Petri nets with read arcs, we developed another tool able to compute the reachability set of any 1-safe net. We then applied it to a set of examples and its unfoldings obtained through our tool. In all the cases the reachability set of the example and its unfolding was the same.

5 Conclusions and future work

In this work, we address the problem of unfolding 1-safe contextual nets. In particular, we tackle the implementation of an unfolder that generates a complete and finite prefix following the abstract unfolding method specified in [1]. Our work constitutes the first implementation of such method, and its development required attention to practical as well as theoretical matters.

On the practical side, we have developed a working contextual unfolder. We have proposed data structures to support the manipulation of the notions involved in the unfolding procedure, such as the history graph or the direct asymmetric conflict graph.

On the theoretical arena, we have also provided several contributions. We have suggested a generalization of adequate orders for contextual unfoldings, and subsequently proved how we can still use them to build a finite and complete prefix. A concurrency relation on enriched conditions has also been suggested, but not implemented. This relation has subsequently been characterized so as to allow the relation itself to be updated as the unfolding grows. Similar ideas have successfully been used in Mole, a Petri net (without read arcs) unfolder, which suggests that this could be an interesting avenue.

The implementation of the contextual unfolder showed to be non trivial (4700 lines of C code). With the goal of soundness in mind, and whenever possible, several theoretical notions were translated into an implementation by turning the theoretical notion *as is* into an algorithm. This is remarkably the case of the conflict relation # between histories, where the unfolder spends in average more than 85% of the time. Consequently, we do not reach still the performance of Mole. Further work is required to review certain data structures and algorithms, such as the relation #. We hope

that this will allow our unfolder to treat moderately larger examples than the current ones.

Concerning the applications, Mole has been applied in the domains of model checking and planning [4]. We plan to pursue similar lines of application with our tool.

References

[1] Paolo Baldan, Andrea Corradini, Barbara König, and Stefan Schwoon. McMillan's complete prefix for contextual nets. In *Transactions on Petri Nets and Other Models of Concurrency I*, pages 199–220, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] Javier Esparza and Keijo Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag, March 2008.

[3] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. In *Formal Methods in System Design*, pages 87–106. Springer-Verlag, 1996.

[4] Sarah Hickmott, Jussi Rintanen, Sylvie Thiébaux, and Lang White. Planning via petri net unfolding. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1904–1911, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[5] K. L. McMillan. A technique of state space search based on unfolding. *Form. Methods Syst. Des.*, 6(1):45–65, 1995.

[6] Kenneth L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *CAV '92: Proceedings of the Fourth International Workshop on Computer Aided Verification*, pages 164–177, London, UK, 1993. Springer-Verlag.

[7] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains. In *Proceedings of the International Sympoisum on Semantics of Concurrent Computation*, pages 266–284, London, UK, 1979. Springer-Verlag.

[8] PEP Project. PEP homepage. [Online; accessed 15-August-2010]: `http://theoretica.informatik.uni-oldenburg.de/~pep/`.

[9] Stefan Schwoon. Mole – an unfolder for Petri Nets. [Online; accessed 15-August-2010]: `http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/`.

A Proofs

We provide in this Appendix proofs of the statements presented in previous sections. For each proof, we first formulate the statement and then present its proof by *making the same assumptions as those made in the section where the statement initially appears.*

Let us present here some notation whose presentation we had to skip due to the space constraints of this document. We say that a configuration C of the full unfolding $\mathcal{U}_N$ (or enriched prefix $\mathcal{E}_N$ of it), *contains* an enriched event $\langle e, H\rangle$ if $e \in C$ and $H = C[\![e]\!]$.

For any enriched event $\varepsilon = \langle e, H\rangle$ of $\mathcal{U}_N$ (or enriched prefix $\mathcal{E}_N$ of it), we will write ε^H to denote H and ε^e to denote e. In a similar way, for enriched conditions $\rho = \langle c, H\rangle$ of $\mathcal{U}_N$, we will write ρ^H to denote H and ρ^c to denote c.

A.1 A finite and complete prefix of $\mathcal{U}_N$

Theorem 9. *$\mathcal{T}_N$ has a finite number of enriched events and is complete.*

Proof. (Sketch, see Theorems 1 and 2 of [1].) By assumption, N is finite and 1-safe net, and has a finite number of reachable markings. Observe that the number of configurations of $\mathcal{T}_N$ must be finite, since if it were infinite, considering the fact that every configuration is associated to one marking in N, we would have infinite cutoffs in $\mathcal{T}_N$, a contradiction. As the number of configurations is finite, the number of enriched events must be finite.

We say that a configuration C contains an enriched event $\langle e, H_e\rangle$ if $e \in C$ and $H_e = C[\![e]\!]$. According to this definition, it is possible to prove that, for any configuration of $\mathcal{U}_N$ we can find a configuration free of cutoffs. That configuration must be present in $\mathcal{T}_N$. Since $\mathcal{U}_N$ is complete, we can conclude that $\mathcal{T}_N$ is also complete. □

A.2 Direct asymmetric conflict

Proposition 12. *For any pair of configurations $C_1, C_2 \in \mathsf{Conf}(\mathcal{U}_N)$, we have $C_1 \sqsubseteq C_2$ iff $C_1 \sqsubseteq_\uparrow C_2$.*

Proof. Assume $\neg(C_1 \sqsubseteq C_2)$. We prove that $\neg(C_1 \sqsubseteq_\uparrow C_2)$. As $\neg(C_1 \sqsubseteq C_2)$, then either $C_1 \not\subseteq C_2$ or there exist some $e_1 \in C_1$ and some $e_2 \in C_2 \setminus C_1$ verifying $e_2 \nearrow e_1$. In the first case, we have $\neg(C_1 \sqsubseteq_\uparrow C_2)$ by definition. In the second, notice that $e_2 \uparrow e_1$ also holds, since if it were the case that $e_2 < e_1$, the fact that $e_1 \in C_1$ and that C_1 is a configuration (and thus, causally closed), would imply that $e_2 \in C_1$, which is a contradiction to $e_2 \in C_2 \setminus C_1$. As $e_2 \nearrow e_1$ and $e_2 \not< e_1$, we have $e_2 \uparrow e_1$. Therefore, $\neg(C_1 \sqsubseteq_\uparrow C_2)$.

Assume now that $\neg(C_1 \sqsubseteq_\uparrow C_2)$. We prove that $\neg(C_1 \sqsubseteq C_2)$. If $\neg(C_1 \sqsubseteq_\uparrow C_2)$, then either $C_1 \not\subseteq C_2$ or there exist some $e_1 \in C_1$ and some $e_2 \in C_2 \setminus C_1$ verifying $e_2 \uparrow e_1$. In the first case, we have $\neg(C_1 \sqsubseteq C_2)$ by definition. In the second, since $e_2 \uparrow e_1$ implies $e_2 \nearrow e_1$, we also have $\neg(C_1 \sqsubseteq C_2)$. □

A.3 History graph

In order to prove Lemma 14 we need to first prove that the edge relation of $\mathcal{H}_N$ is acyclic:

Lemma 18. *The edge relation $\rightarrow$ of any given history graph $\mathcal{H}_N = (V, \rightarrow)$ is acyclic.*

Proof. For an argument by contradiction, assume that we can find a loop in the relation $\rightarrow$ using enriched events of V. Let $\langle e_1, H_1\rangle \rightarrow \langle e_2, H_2\rangle \rightarrow \ldots \rightarrow \langle e_n, H_n\rangle$ be $n-1$ enriched events of V such that $\langle e_1, H_1\rangle = \langle e_n, H_n\rangle$. The definition of $\rightarrow$ assures that for any $\langle e_i, H_i\rangle$, with $1 \leq i < n$ it holds that $e_{i+1} \in H_i$, and that $H_{i+1} \subseteq H_i$, and that $e_{i+1} \uparrow e_i$. This implies that we have a cycle in the relation $\uparrow$ using events $e_1, \ldots, e_{n-1}$. It also implies that $e_i \in H_1$ for $1 \leq i < n$. These two facts together say that we can find a cycle of the relation $\uparrow$ restricted to H_1, which is a contradiction, since this means that we can find a loop in the relation $\nearrow$ restricted to H_1.

Therefore, relation such cycle in relation $\rightarrow$ cannot be found, and $\rightarrow$ is acyclic. □

Lemma 14. *Given $\mathcal{H}_N = (V, \rightarrow)$ and $\varepsilon = \langle e, H\rangle \in V$, we have $H = \{e' \in T' \mid \varepsilon \rightarrow^* \langle e', H'\rangle\}$.*

Proof. Due to the fact that $\mathcal{H}_N$ is acyclic (Lemma 18), we know that $\rightarrow$ is well founded. We prove the statement by structural induction on the edge relation $\rightarrow$.

Base Node ε has no outgoing edge. We have to prove that $H = \{e\}$. For an argument by contradiction, assume that there is at least one enriched event $e_1 \in H \setminus \{e\}$. As $e_1 \in H$ and $e_1 \neq e$, we have $e_1 \nearrow_H^+ e$. This implies that we can find some event $e_2 \in H$ that is in asymmetric conflict to e, $e_2 \nearrow e$ and such that $e_2 \neq e$. Now we claim that we can find some other event $e_3 \in H$ verifying $e_3 \uparrow e$ and $e_3 \neq e$. Indeed, if $e_2 \uparrow e$, we can take e_2 as such e_3. Otherwise, if $\neg(e_2 \uparrow e)$, we know that $e_2 < e$ and, since H is causally closed, we can find some $e_3 \in H$ verifying $e_2 < e_3 < e$ and $e_3 \uparrow e$. From the definition of enriched prefix, and due to the fact that H is a configuration of $\mathcal{E}_N$, it must hold $H[\![e_3]\!] \in \chi(e_3)$. For this reason, $\langle e_3, H[\![e_3]\!]\rangle$ must be a node of $\mathcal{H}_N$, that is, $\langle e_3, H[\![e_3]\!]\rangle \in V$. Since, by construction, $e_3 \in H$, and $e_3 \uparrow e$, it must hold that $\varepsilon = \langle e, H\rangle \rightarrow \langle e_3, H[\![e_3]\!]\rangle$. This is a contradiction to the assumption that node ε has no outgoing edge. Therefore, $H = \{e\}$.

Step Assume that the statement holds for any $\varepsilon' = \langle e', H'\rangle$ such that $\varepsilon \rightarrow \varepsilon'$. We can write the next development:

$$\{e' \in T' \mid \varepsilon \rightarrow^* \langle e', H'\rangle\} = \{e\} \cup \{e' \in T' \mid \varepsilon \rightarrow^+ \langle e', H'\rangle\}$$

Now, consider the set $D \stackrel{def}{=} \{e' \in H \mid e' \uparrow e\}$. By the definition of $\mathcal{H}_N$, we can see that if $\varepsilon \rightarrow \langle e', H[\![e']\!]\rangle$ is an edge of $\mathcal{H}_N$, then $e' \in D$ holds. This let us write the next development:

$$\{e\} \cup \{e' \in T' \mid \varepsilon \rightarrow^+ \langle e', H'\rangle\} = \{e\} \cup \bigcup_{e' \in D} \{e'' \in T' \mid \langle e', H[\![e']\!]\rangle \rightarrow^* \langle e'', H''\rangle\}$$

By induction hypothesis, we know that the lemma holds for all nodes $\varepsilon' = \langle e', H' \rangle$ such that $\varepsilon \to \varepsilon'$. In other words, we know that for any $e' \in D$, we have $H[\![e']\!] = \{e'' \in T' \mid \langle e', H[\![e']\!] \rangle \to^* \langle e'', H'' \rangle\}$. Again, this allows us to write

$$\{e\} \cup \bigcup_{e' \in D} \{e'' \in T' \mid \langle e', H[\![e']\!] \rangle \to^* \langle e'', H'' \rangle\} = \{e\} \cup \bigcup_{e' \in D} H[\![e']\!]$$

Finally, it is easy to see that

$$\{e\} \cup \bigcup_{e' \in D} H[\![e']\!] = H$$

by means of a subset inclusion in both directions. Direction $\subseteq$ is trivial. Direction $\supseteq$ can be proved by regarding a partition of H in three sets, namely $\{e\}$, D and $(H \setminus \{e\}) \setminus D$. For events in the first two sets, it is trivial to see that the inclusion holds. For any event $e'' \in (H \setminus \{e\}) \setminus D$, it is not difficult to see that $e'' \in H[\![e']\!]$ for some $e' \in D$. □

A.4 Cutoff criterion and adequate orders

In order to provide a proof for Theorem 16, we need to first provide some two new lemmas: Lemma 19 and Lemma 20.

Lemma 19. *$\widehat{\mathcal{T}}_N$ has a finite number of enriched events*

Proof. By contradiction, assume that $\widehat{\mathcal{T}}_N$ has infinite enriched events. As any history of an enriched event is a configuration, and, in particular, a *finite* configuration, we also know that the number of finite configurations of $\mathcal{T}_N$ is infinite. In the sequel, we see that under these assumptions we can use König's lemma to find an infinite sequence of configurations $C_1, C_2, \ldots$ with $C_i \sqsubset C_{i+1}$ and $\mathsf{Mark}(C_1) = \mathsf{Mark}(C_i)$ for $i \geq 1$, yielding to the fact that C_2 is a cutoff, which is a contradiction to the definition of $\mathcal{T}_N$.

We define a graph $G = (V, E)$ whose set of vertices V is the set of finite configurations of $\mathcal{T}_N$ plus the *empty configuration* $\emptyset$, and whose set of edges is a subset of the computational order relation $\sqsubseteq$:

- $V = \mathsf{Conf}(\mathcal{T}_N) \cup \{\emptyset\}$
- $E = \{(C_1, C_2) \in V \times V \mid C_1 \sqsubseteq C_2 \text{ and } |C_1| + 1 = |C_2|\}$

We write $C_1 \to C_2$ for $(C_1, C_2) \in E$. We now prove three properties of G:

G has finite degree. We show that for any C_1 there is only a finite number of configurations C_2 such that $C_1 \to C_2$.

Consider any pair of configurations C_1, C_2 with $C_1 \to C_2$. Then $C_1 \sqsubseteq C_2$ and $|C_1| + 1 = |C_2|$. By definition, $C_1 \sqsubseteq C_2$ implies $C_1 \subseteq C_2$. Then we can write $C_2 = C_1 \cup \{e\}$ for some $e \notin C_1$. From the fact that $C_1 \sqsubseteq C_2$ we also know that $\neg(e_2 \nearrow e_1)$ for all $e_1 \in C_1$ and all $e_2 \in C_2 \setminus C_1$. In particular $\neg(e \nearrow e_1)$ for all $e_1 \in C_1$, so e can fire after any event e_1 of C_1. This, together with the fact

that C_2 is a configuration, implies that ${}^\bullet e \subseteq \mathsf{Cut}(C_1)$. From this, it follows that $e \in \mathsf{Cut}(C_1)^\bullet$.

Now notice that $\mathsf{Cut}(C_1)^\bullet$ is finite, because C_1 is finite and the preset and postset of each event in $\mathcal{U}_N$ is finite (due to our initial assumption about the finiteness of N). So there is, at most, a finite number of *different* events $e \in \mathsf{Cut}(C_1)^\bullet$ and hence a finite number of different sets $C_1 \cup \{e\}$. As we saw previously, any C_2 such that $C_1 \to C_2$ is one of these $C_1 \cup \{e\}$ for $e \in \mathsf{Cut}(C_1)^\bullet$. Hence, for a given C_1, there is at most a finite number of different edges $C_1 \to C_2$ in G.

G is connected. We show by induction on the size of a configuration C that there exists a path from the empty configuration $\emptyset$ to C. In the base case, $|C| = 0$ and consequently $C = \emptyset$. The statement is trivially true. For the step case, assume that we can find a path for configurations of up to size n, or more formally, that $\emptyset \to^* C$ if $|C| \leq n$ and let C' be a configuration of size $n+1$. From C' we can extract a event e maximal w.r.t. the relation $\nearrow$. This is always possible because C' is finite and there is no loops in $\nearrow$ restricted to C'. As there exists no other $e' \in C'$ with $e \nearrow e'$, we can assure that $C' \setminus \{e\}$ is a configuration. Furthermore, it is a configuration of size n for which the induction hypothesis applies. Finally, notice that $C' \setminus \{e\} \sqsubseteq C'$, because $C' \setminus \{e\} \subseteq C'$ and the maximality of e. Consequently, $C' \setminus \{e\} \to C'$.

G has an infinite number of vertices. By hypothesis, we know that there is an infinite number of configurations in $\mathcal{T}_N$. Each configuration is a vertex in G.

In the light of these properties, König's lemma guarantees the existence in G of an infinite sequence of configurations $(C_i)_{i\geq 1}$ such that $C_1 = \emptyset$ and $C_i \to C_{i+1}$. By hypothesis, N is n-bounded, so the number of reachable markings in N is finite. As each configuration C_i has an associated marking $\mathsf{Mark}(C_i)$, this implies that from $(C_i)_{i\geq 1}$ we can extract an infinite subsequence of configurations $C_{i_1}, C_{i_2}, \ldots$ marked with the same marking, that is, verifying:

- (subsequence) $i_j < i_{j+1}$ for $j \geq 1$, and
- (same marking) $\mathsf{Mark}(C_{i_1}) = \mathsf{Mark}(C_{i_j})$ for $j \geq 2$, and

Consider now the configurations C_{i_1} and C_{i_2}. By construction, we know that $C_{i_1} \to^* C_{i_2}$. This, together with the fact that relation $\sqsubseteq$ is transitive, implies that $C_{i_1} \sqsubseteq C_{i_2}$ and by Definition 15, that $C_{i_1} \prec C_{i_2}$. But, by construction, we know that $\mathsf{Mark}(C_{i_1}) = \mathsf{Mark}(C_{i_2})$, so C_{i_2} is a cutoff. This is a contradiction to the fact that $\mathcal{T}_N$ is free of cutoffs.

Hence, the number of enriched events in $\mathcal{T}_N$ is finite. □

Lemma 20. *If a finite configuration C of $\mathcal{U}_N$ contains at least one adequate cutoff, then there exists another configuration C' such that $\mathsf{Mark}(C) = \mathsf{Mark}(C')$ and $C' \prec C$.*

Proof. Assume that C contains the adequate cutoff $\langle e, H_e\rangle$, for some $e \in C$ and $H_e = C[\![e]\!]$. Then, by definition, we know that either $\mathsf{Mark}(H_e) = m'$, with m' the initial marking of $\mathcal{U}_N$, or there exists another enriched event $\langle e', H_{e'}\rangle$ such that $\mathsf{Mark}(H_e) =$

$\mathsf{Mark}(H_{e'})$ and $H_{e'} \prec H_e$. In both cases, we regard history H defined as $H \stackrel{def}{=} \emptyset$ in the first case and $H \stackrel{def}{=} H_{e'}$ in the second case. Notice that in any case, $H \prec H_e$ (if $H = \emptyset$, then $H \sqsubset H_e$ and by definition, $H \prec H_e$).

As $\mathsf{Mark}(H_e) = \mathsf{Mark}(H)$, it is possible to find an extension of H firing exactly the same transitions as those labeling the set $C \setminus H_e$. In other words, it must be possible to find an extension E of H isomorphic to $C \setminus H_e$, such that $\mathsf{Mark}(H_e \cup (C \setminus H_e)) = \mathsf{Mark}(H \cup E)$. This follows from the fact that $\mathcal{U}_N$ unfolds *as much as possible* and can be proved by induction on the size of the set $C \setminus H_e$:

Base $|C \setminus H_e| = 1$. Assume that $C \setminus H_e = \{e'\}$. As $\mathsf{Mark}(H_e) = \mathsf{Mark}(H)$, then the transition t labeling e' is enabled at $\mathsf{Mark}(H)$ and, by the definition of full unfolding, there exists an event e'' labeled by t such that $H \cup \{e''\}$ is a configuration. Hence, $\{e''\}$ is an extension of H isomorphic to $C \setminus H_e$.

Step Assume we can find extensions of H isomorphic to $C \setminus H_e$ up to size n. We show that it is also possible for size $n+1$. Assume that $|C \setminus H_e| = n+1$ and consider any maximal event e' of $C \setminus H_e$ w.r.t. the relation $\nearrow$ (it must exists because $\nearrow$ is acyclic in C). As e' can fire after any event in $C \setminus H_e$, and therefore any event in C, we know that $C \setminus \{e'\}$ is a configuration and that the set $(C \setminus H_e) \setminus \{e'\}$ is an extension of H_e of size n. Then, the induction hypothesis applies and says that we can find an extension E of H, isomorphic to $(C \setminus H_e) \setminus \{e'\}$ such that $\mathsf{Mark}(C \setminus \{e'\}) = \mathsf{Mark}(H \cup E)$. But now notice that the transition t labeling e' is enabled at $\mathsf{Mark}(H \cup E)$ and, therefore there exists an event e'' labeled by t such that $H \cup E \cup \{e''\}$ is a configuration and $E \cup \{e''\}$ is an extension of H isomorphic to $C \setminus H_e$.

We claim now that $C' \stackrel{def}{=} H \cup E$ is the configuration we are searching for. To see this, we just have to verify that $C' \prec C$. We do it using the third condition of Definition 15. Indeed, H_e and H are configurations with the same marking; by hypothesis $H_e \sqsubset C$ and by construction $H \sqsubset H \cup E$ and E is isomorphic to $C \setminus H_e$. Hence $H \cup E \prec C$. □

Theorem 16. *$\widehat{\mathcal{T}}_N$ has a finite number of enriched events and is complete.*

Proof. That $\widehat{\mathcal{T}}_N$ has a finite number of enriched events is proved by Lemma 19. We prove now that, making use of Lemma 20, that we can show completeness of $\widehat{\mathcal{T}}_N$.

We know after Proposition 6 that $\mathcal{U}_N$ is a complete enriched prefix. Therefore, for any marking m of N we can find a configuration $C \in \mathsf{Conf}(\mathcal{U}_N)$ with $\mathsf{Mark}(C) = m$. We show now that for such configuration C we can find another configuration C' with the same marking and free of adequate cutoffs that, consequently, belongs to $\mathcal{T}_N$.

Let $C \in \mathsf{Conf}(\mathcal{U}_N)$ be a configuration of the full unfolding. If C is free of cutoffs, then all enriched events $\langle e, C[\![e]\!]\rangle$ are enriched events of $\widehat{\mathcal{T}}_N$ and therefore C is a configuration of $\widehat{\mathcal{T}}_N$.

So, let us assume that C contains a cutoff. By Lemma 20 we can find another configuration $C_1 \in \mathsf{Conf}(\mathcal{U}_N)$ with $\mathsf{Mark}(C) = \mathsf{Mark}(C_1)$ and $C_1 \prec C$. If C_1 still contains cutoffs, we can apply again Lemma 20 and find another configuration $C_2 \prec C_1$ with $\mathsf{Mark}(C_2) = \mathsf{Mark}(C)$. Furthermore, as $\prec$ is well founded, the number of times we have to apply Lemma 20 in order to find a configuration C_n free of cutoffs is finite,

otherwise we would either have an infinite decreasing sequence of configurations w.r.t. the order $\prec$, or a counterexample for Lemma 20. $\square$

A.5 Optimizations

The goal of this section is providing a proof for Theorem 17. For the sake of simplicity in the following developments, we first characterize the concurrency relation in terms of the $\nearrow$ relation.

Remark 21. *The statement $\langle H, c\rangle \parallel \langle H', c'\rangle$ is equivalent to the conjunction of the next four statements:*

1. $\neg(\exists e_1 \in H,\, \exists e_2 \in H' \setminus H,\, e_2 \nearrow e_1)$
2. $\neg(\exists e_1 \in H',\, \exists e_2 \in H \setminus H',\, e_2 \nearrow e_1)$
3. $\neg(\exists e_1 \in H,\, c' \in {}^\bullet e_1)$
4. $\neg(\exists e_1 \in H',\, c \in {}^\bullet e_1)$

More intuitively, Remark 21 says that $\langle H, c\rangle \parallel \langle H', c'\rangle$ holds if and only if H can evolve to $H \cup H'$ (1), H' can evolve to $H \cup H'$ (2) and none of any histories consumes the condition generated by the other history (3 and 4). Note that the conjunction of conditions 1 and 2 is equivalent to $\neg(H \# H')$ while the conjunction of conditions 3 and 4 is equivalent to $c, c' \in \mathsf{Cut}(H \cup H')$.

Let us now provide the technical lemmas Lemma 22 and Lemma 23.

Lemma 22. *Let $\mathcal{E}_N = \langle \mathcal{P}_N, \chi\rangle$ be an enriched prefix and $H, H' \in \chi(e)$ be two histories of some event e of $\mathcal{P}_N$. If $H \neq H'$, then $H \# H'$.*

Proof. As both histories are histories for event e, it holds that $e \in H$ and $e \in H'$. As $H \neq H'$, it holds that $H \setminus H' \neq \emptyset$. This means that we can find some event $e_1 \in H \setminus H'$ in H and not in H'. Due to the fact that H is a history of e, we can assure that $e_1 \nearrow_H^* e$. Taking into account that $e_1 \notin H'$ and that $e \in H'$, we can be sure that $e_1 \neq e$, which entails that $e_1 \nearrow_H^+ e$ holds. In turn, this implies that we can find events $e_2, \ldots, e_n \in H$ such that $n \geq 2$, and $e = e_n$, and $e_1 \nearrow e_2 \nearrow \ldots \nearrow e_{n-1} \nearrow e_n$. Additionally, $e_1 \in H \setminus H'$ and $e_n \in H'$. It is now easy to see that there exists some e_i, with $1 \leq i < n$, verifying $e_i \in H \setminus H'$ and $e_{i+1} \in H'$. This implies that $H \# H'$ $\square$

Lemma 23. *Let H, H' be two histories of $\mathcal{E}_N = \langle \mathcal{P}_N, \chi\rangle$, with $H' \prec H$, and $H \in \chi(e)$ and $e \in H'$. Then $H \# H'$.*

Proof. We claim that $H \not\sqsubseteq H'$. Indeed, if it were the case that $H \sqsubseteq H'$, and provided that any adequate order $\prec$ is antisymmetric, we would have $H \prec H'$, a contradiction to the hypothesis $H' \prec H$. So $H \not\sqsubseteq H'$ holds. Making use of the definition of $\sqsubseteq$ (page 8), one can write the next equivalence:

$$H \not\sqsubseteq H' \iff H \not\subseteq H' \vee \exists e_1 \in H,\, \exists e_2 \in H' \setminus H,\, e_2 \nearrow e_1$$

As $H \not\sqsubseteq H'$ holds, the right-hand side also holds. We show that both sides of the disjunction in the right-hand size implies $H \# H'$.

Clearly, $\exists e_1 \in H, \exists e_2 \in H' \setminus H, e_2 \nearrow e_1$ implies $H \# H'$. On the other hand, $H \not\subseteq H'$ implies that we can find some event $e'' \in H \setminus H'$. Now notice that, as $e'' \in H$, it holds that $e'' \nearrow_H^* e$. Event e and e'' must be different because $e \in H'$ by hypothesis and $e'' \notin H'$. Consequently it also holds that $e'' \nearrow_H^+ e$. This means that we can find a finite number of events $e_1, \ldots, e_n \in H$ with $e'' = e_1 \nearrow e_2 \nearrow \ldots \nearrow e_n = e$. It is easy to see that there must exist some pair of events e_l, e_{l+1} for some $1 \leq l < k$ with $e_l \in H \setminus H'$ and $e_{l+1} \in H'$. By construction, $e_l \nearrow e_{l+1}$ and hence $H \# H'$. □

Theorem 17. *Let $\rho = \langle H, c \rangle$, $\rho' = \langle H', c' \rangle$ be two different enriched conditions of $\mathcal{E}_N$ such that $H \in \chi(e)$, $H' \in \chi(e')$ and $H' \prec H$ for an adequate order $\prec$. We have the equivalence*

$$\rho \parallel \rho' \iff \bigwedge_{\rho_i \in {}^{\bullet\bullet}\rho} \rho_i \parallel \rho' \wedge \bigwedge_{\sigma_j \in \underline{{}^{\bullet}\rho}} \sigma_j \parallel \rho' \wedge (\rho' \notin {}^{\bullet\bullet}\rho) \wedge \neg \exists e'' \in H' \setminus H, \underline{e''} \cap {}^{\bullet}e \neq \emptyset$$

Proof. Let ${}^{\bullet\bullet}\rho = \{\rho_1, \ldots, \rho_n\}$ and $\underline{{}^{\bullet}\rho} = \{\sigma_1, \ldots, \sigma_m\}$. We let variable $\rho_i = \langle H_i, c_i \rangle$ to range the set ${}^{\bullet\bullet}\rho$ and variable $\sigma_j = \langle H_j, c_j \rangle$ to range the set $\underline{{}^{\bullet}\rho}$, with $1 \leq i \leq n$ and $1 \leq j \leq m$. For any ρ_i and σ_j, we let $H_i \in \chi(e_i)$ and $H_j \in \chi(e_j)$. Note also that, by definition, $H_i = H[\![e_i]\!]$ and that $H_j = H[\![e_j]\!]$. It is easy also to see that for any pair $\rho_1, \rho_2 \in {}^{\bullet\bullet}\rho \cup \underline{{}^{\bullet}\rho}$ we have $\rho_1 \parallel \rho_2$. Finally note that, also by construction, $H = \{e\} \cup \bigcup_{\rho_i \in {}^{\bullet\bullet}\rho} \rho_i^H \cup \bigcup_{\sigma_j \in \underline{{}^{\bullet}\rho}} \sigma_j^H$.

We prove in the sequel both directions of the co-implication. From left to right, we prove that the hypothesis implies each one of the conjunctions.

(a) $\rho \parallel \rho' \implies \bigwedge_{\rho_i \in {}^{\bullet\bullet}\rho} \rho_i \parallel \rho'$ It is enough to prove that $\rho \parallel \rho' \implies \rho_i \parallel \rho'$ for *some* ρ_i. As we do not make any assumption on ρ_i, the proof will be valid for *all* ρ_i.

Assume that $\rho \parallel \rho'$ and that, for a proof by contradiction, $\neg(\rho_i \parallel \rho')$ for some $\rho_i \in {}^{\bullet\bullet}\rho$. Then we know that at least one of the four statements in Remark 21 must be false when regarding ρ_i and ρ'. We proceed by cases:

1. Assume that there exist events $e_1 \in H_i$ and $e_2 \in H' \setminus H_i$ with $e_2 \nearrow e_1$. Two cases are possible: either $e_2 \in H$ or $e_2 \notin H$. If $e_2 \notin H$, as $H_i \subseteq H$, we have that $e_1 \in H$, $e_2 \in H' \setminus H$ and $e_2 \nearrow e_1$, which implies $H \# H'$. This is a contradiction to $\rho \parallel \rho'$. So, let us assume that $e_2 \in H$. As $H = \{e\} \cup \bigcup_{\rho_i \in {}^{\bullet\bullet}\rho} \rho_i^H \cup \bigcup_{\rho_j \in \underline{{}^{\bullet}\rho}} \rho_j^H$, several sub-cases are possible:
   - $e_2 \in \rho_{i'}^H$ for some $\rho_{i'} \in {}^{\bullet\bullet}\rho$ with $i' \neq i$. Again, we have $e_1 \in H_i$, $e_2 \in \rho_{i'}^H \setminus H_i$ and $e_2 \nearrow e_1$. This implies $\rho_{i'}^H \# H_i$, which is a contradiction to $\rho_i \parallel \rho_{i'}$.
   - $e_2 \in \sigma_j^H$ for some $\sigma_j \in \underline{{}^{\bullet}\rho}$. Likewise, we have $e_1 \in H_i$, $e_2 \in \sigma_j^H \setminus H_i$, $e_2 \nearrow e_1$ and $\sigma_j^H \# H_i$, which is a contradiction to $\rho_i \parallel \sigma_j$.
   - $e_2 = e$. By definition Definition 5, we have $e_1 \nearrow_{H_i}^* e_i$ (recall that $H_i \in \chi(e_i)$). By construction of H, we have $e_i \nearrow e$. By hypothesis, we have $e_2 = e$ and also by hypothesis we have $e_2 \nearrow e_1$. This leads to the cycle $e_1 \nearrow_{H_i}^* e_i \nearrow e = e_2 \nearrow e_1$ in the asymmetric conflict relation. Now notice that $e_1, e_i, e \in H$, and that $H_i \subseteq H$. Therefore, we have a loop in the asymmetric conflict relation restricted to the history H, which is a contradiction to the fact that H is a history.

2. Assume that there exist $e_1 \in H'$ and $e_2 \in H_i \setminus H'$ with $e_2 \nearrow e_1$. Then, as $H_i \subseteq H$, we have that $e_2 \in H \setminus H'$. This together with $e_1 \in H'$ and $e_2 \nearrow e_1$ implies $H \# H'$, which is a contradiction to $\rho \parallel \rho'$.
3. Assume that there exists $e_1 \in H_i$ such that $c' \in {}^\bullet e_1$. Intuitively, this means that H_i consumes c'. As $H_i \subseteq H$, also H consumes c', which leads to a contradiction of $\rho \parallel \rho'$.
4. Assume that there exists $e_1 \in H'$ such that $c_i \in {}^\bullet e_1$. We have two cases, either $e_1 = e$ or $e_1 \neq e$. If we assume that $e_1 \neq e$, then we have that ${}^\bullet e \cap {}^\bullet e_1 \neq \emptyset$. This implies that $e \nearrow e_1$ and that $e_1 \nearrow e$. In turn, it implies that $e_1 \notin H$ (otherwise we would have a loop in the relation $\nearrow$ restricted to H). So assuming that $e_1 \neq e$ would lead to conclude that $e_1 \in H' \setminus H$ and that $e_1 \nearrow e$, with $e \in H$, which implies that $H \# H'$, a contradiction to $\rho \parallel \rho'$.

   Therefore, let us assume that $e_1 = e$ and, consequently, that $e \in H'$. As, by hypothesis $H' \prec H$, we can apply Lemma 23 and conclude that $H \# H'$, a contradiction to $\rho \parallel \rho'$.

(b) $\rho \parallel \rho' \implies \bigwedge_{\sigma_j \in \underline{{}^\bullet \rho}} \sigma_j \parallel \rho'$ As in (a), it is still enough to prove that the statement holds for just one σ_j. In particular, we will prove that $\rho \parallel \rho' \implies \sigma_j \parallel \rho'$ for some $\rho_j \in \underline{{}^\bullet \rho}$. As we make no assumption about σ_j, the argument will be valid for *all* σ_j.

We reason by contradiction. Assume that both $\rho \parallel \rho'$ and $\neg(\sigma_j \parallel \rho')$ hold, for some $\sigma_j \in \underline{{}^\bullet \rho}$. As $\neg(\sigma_j \parallel \rho')$, we know that at least one of the four statements in Remark 21 must not hold. In the following, we see that we can find a contradiction in each case.

1. Assume that there exist events $e_1 \in H_j$ and $e_2 \in H' \setminus H_j$ with $e_2 \nearrow e_1$. As in (a.1), two cases are possible: either $e_2 \in H$ or $e_2 \notin H$. If $e_2 \notin H$, as $H_j \subseteq H$, we have that $e_1 \in H$, $e_2 \in H' \setminus H$ and $e_2 \nearrow e_1$, which implies $H \# H'$. This is a contradiction to $\rho \parallel \rho'$. So, let us assume that $e_2 \in H$. As $H = \{e\} \cup \bigcup_{\rho_i \in {}^\bullet\bullet \rho} \rho_i^H \cup \bigcup_{\sigma_j \in \underline{{}^\bullet \rho}} \sigma_j^H$, several sub-cases are possible:
   i. $e_2 \in \sigma_{j'}^H$ for some $\sigma_{j'} \in \underline{{}^\bullet \rho}$ with $j' \neq j$. Again, we have $e_1 \in H_j$, $e_2 \in \sigma_{j'}^H \setminus H_j$ and $e_2 \nearrow e_1$. This implies $\sigma_{j'}^H \# H_j$, which is a contradiction to $\sigma_j \parallel \sigma_{j'}$.
   ii. $e_2 \in \rho_i^H$ for some $\rho_i \in {}^\bullet\bullet \rho$. Likewise, we have $e_1 \in H_j$, $e_2 \in \rho_i^H \setminus H_j$, $e_2 \nearrow e_1$ and $\rho_i^H \# H_j$, which is a contradiction to $\sigma_j \parallel \rho_i$.
   iii. $e_2 = e$. Recall that $H_j \in \chi(e_j)$. By definition Definition 5, we have $e_1 \nearrow_{H_j}^* e_j$. By construction of H, we have $e_j \nearrow e$. By hypothesis, we have $e_2 = e$ and also by hypothesis we have $e_2 \nearrow e_1$. This leads to the cycle $e_1 \nearrow_{H_j}^* e_j \nearrow e = e_2 \nearrow e_1$ in the asymmetric conflict relation. Now notice that $e_1, e_j, e \in H$, and that $H_j \subseteq H$. Therefore, we have a loop in the asymmetric conflict relation restricted to the history H, which is a contradiction to the fact that H is a history.
2. Assume that there exist $e_1 \in H'$ and $e_2 \in H_j \setminus H'$ with $e_2 \nearrow e_1$. The same argument as in (a.2) is still applicable here, changing H_i by H_j.
3. Assume that there exists $e_1 \in H_j$ such that $c' \in {}^\bullet e_1$. Same argument as in (a.3), substituting H_i by H_j.

4. Assume that there exists $e_1 \in H'$ such that $c_j \in {}^\bullet e_1$. We then know that $\underline{e} \cap {}^\bullet e_1 \neq \emptyset$. As we deal with nets in which for any event e'' it is not the case that $\underline{e''} \cap {}^\bullet e'' \neq \emptyset$, we have to assume that $e \neq e_1$. Notice also that $e \nearrow e_1$. We claim now that $e \in H'$. By contradiction, if it were the case that $e \notin H'$, we would have that $e \in H \setminus H'$, $e_1 \in H'$ and $e \nearrow e_1$, and consequently $H \# H'$.
   Therefore we know that $e \in H'$. By hypothesis we also know that $H' \prec H$. Under this assumptions we can apply Lemma 23 and conclude that $H \# H'$, a contradiction to $\rho \parallel \rho'$.

(c) $\rho \parallel \rho' \implies \rho' \notin {}^{\bullet\bullet}\rho$ It is easy to see that it cannot be the case that $\rho' \in {}^{\bullet\bullet}\rho$ holds if we assume $\rho \parallel \rho'$. Indeed, if $\rho' \in {}^{\bullet\bullet}\rho$ we have that $\rho' = \rho_i = (H_i, c_i)$ for some $\rho_i \in {}^{\bullet\bullet}\rho$. But notice that event $c_i \in {}^\bullet e$, that is, event e consumes c_i. This implies that the third statement of Remark 21 doesn't hold with regard to ρ and ρ' and we have $\neg(\rho \parallel \rho')$, a contradiction.

(d) $\rho \parallel \rho' \implies \neg\exists e'' \in H' \setminus H, \underline{e''} \cap {}^\bullet e \neq \emptyset$ Assume, for an argument by contradiction, that it is the case that there exists $e'' \in H' \setminus H$ such that $\underline{e''} \cap {}^\bullet e \neq \emptyset$. In consequence, we have $e'' \nearrow e$, with $e \in H$ and $e'' \in H' \setminus H$, which implies $H \# H'$, a contradiction to $\rho \parallel \rho'$.

(e) $\rho \parallel \rho' \impliedby \bigwedge_{\rho_i \in {}^{\bullet\bullet}\rho} \rho_i \parallel \rho' \wedge \bigwedge_{\rho_j \in \underline{{}^\bullet \rho}} \rho_j \parallel \rho' \wedge (\rho' \notin {}^{\bullet\bullet}\rho) \wedge \neg\exists e'' \in H' \setminus H, \underline{e''} \cap {}^\bullet e \neq \emptyset$ We prove now the opposite direction of the theorem. We assume the right-hand side of the implication and the negation of the left-hand side. As $\neg(\rho \parallel \rho')$, one of the statements of Remark 21 must be false:

1. Assume that there exist events $e_1 \in H$ and $e_2 \in H' \setminus H$ with $e_2 \nearrow e_1$. Recall that $H = \{e\} \cup \bigcup_{\rho_i \in {}^{\bullet\bullet}\rho} \rho_i^H \cup \bigcup_{\sigma_j \in \underline{{}^\bullet \rho}} \sigma_j^H$. We regard e_1 and reason by cases:
   - Assume that $e_1 \in H_i$ for some $\rho_i \in {}^{\bullet\bullet}\rho$. As $H_i \subseteq H$, we still have $e_2 \in H' \setminus H_i$, and hence $H_i \# H'$, a contradiction to $\rho_i \parallel \rho'$.
   - Assume that $e_1 \in H_j$ for some $\sigma_j \in \underline{{}^\bullet \rho}$. In the same way, we can see that $H_j \# H'$, a contradiction.
   - Finally, assume that $e_1 = e$ and, consequently, $e_2 \nearrow e$. Definition 2 provides us three cases:
     - Assume that ${}^\bullet e_2 \cap {}^\bullet e \neq \emptyset$. Under this assumption, $e_2 \in H'$ clearly consumes c_i for one $\rho_i \in {}^{\bullet\bullet}\rho$, which implies that $\neg(\rho' \parallel \rho_i)$, a contradiction.
     - Assume that $e_2 < e$. We know that H is a history, and by definition contains all events $e'' < e$. As e_2 is one such event, we have $e_2 \in H$, which is a contradiction to the assumption $e_2 \in H' \setminus H$.
     - Assume that $\underline{e_2} \cap {}^\bullet e \neq \emptyset$. This is a contradiction to the last conjunction in hypothesis of the statement that we are proving.
2. Assume that there exist events $e_1 \in H'$ and $e_2 \in H \setminus H'$ with $e_2 \nearrow e_1$. Using the same arguments as in (e.1), we can immediately discard the cases where $e_2 \in H_i$ for some $\rho_i \in {}^{\bullet\bullet}\rho$ or $e_2 \in H_j$ for some $\sigma_j \in \underline{{}^\bullet \rho}$. We assume, hence, that $e_2 = e$. Definition 2 gives us three cases to examine in the relation $e \nearrow e_1$:
   - Assume that ${}^\bullet e \cap {}^\bullet e_1 \neq \emptyset$. Then event $e_1 \in H'$ consumes c_i for some $\rho_i \in {}^{\bullet\bullet}\rho$, which leads to the contradiction $\neg(\rho_i \parallel \rho)$.

- Assume that $e < e_1$. As $e_1 \in H'$ and H' is a history, we should have $e \in H'$, while by hypothesis $e = e_2 \in H \setminus H'$ and hence $e \notin H'$.
- Assume that $\underline{e} \cap {}^\bullet e_1 \neq \emptyset$. Then $c_j \in {}^\bullet e_1$ for some $\sigma_j \in {}^\bullet\underline{\rho}$. As $e_1 \in H'$, we have that $\neg(\sigma_j \parallel \rho')$. This is a contradiction.

3. Assume that there exists $e_1 \in H$ such that $c' \in {}^\bullet e_1$. If we assume $e_1 \in H_i$ for some $\rho_i \in {}^{\bullet\bullet}\rho$, we will find the contradiction $\neg(\rho_i \parallel \rho')$. Similarly, if we assume $e_1 \in H_j$ for some $\sigma_j \in {}^\bullet\underline{\rho}$ we will reach the contradiction $\neg(\sigma_j \parallel \rho')$. So the only case that we examine is when $e_1 = e$. If $c' \in {}^\bullet e$, we have that $c' = c_i$ for some $\rho_i \in {}^{\bullet\bullet}\rho$. As $|{}^\bullet c''| = 1$ for any c'', we have that $H_i \in \chi(e'')$ iff $H' \in \chi(e'')$ for some e''. Intuitively, this means that H_i and H' are histories for the same event. Furthermore, they are different by hypothesis ($\rho' \notin {}^{\bullet\bullet}\rho$ is one of the hypothesis). Under this assumptions, we can apply Lemma 22 and conclude that $H' \# H_i$, which is a contradiction.
4. Assume that there exists $e_1 \in H'$ such that $c \in {}^\bullet e_1$. Then $e \in H'$, and H' consumes any c_i for $\rho_i \in {}^{\bullet\bullet}\rho$, which is a contradiction.

□

B Algorithms

In this section we present the pseudocode of the algorithms used to compute the procedure pe_update of Algorithm 1. Algorithm 2 presents this procedure. As the reader can see, its operation is divided in three steps, namely, procedures pe_update_context, pe_update_existing and pe_update_new, presented respectively in Algorithm 3, Algorithm 5 and Algorithm 4. What follows is a brief intuitive description of the rationale for this division.

Procedure pe_update_context, when called on the argument $\langle e, H\rangle$ returns the set of all possible extensions of $\langle e', H'\rangle$ induced by $\langle e, H\rangle$ such that e' consumes a condition in the context of e. All the possible enriched events that it can return are built on the base of events e' that are already present in $\mathcal{E}_N$ (in T'). No new occurrence of a transition from N is returned. To compute new histories for such events e', pe_update_context uses the histories already present in $\mathcal{E}_N$ for e' and tries to *extend* each one of them with H. This already provides a gain in performance over the general method presented in §3.1, since we can avoid here a systematic enumeration of all the histories generating conditions ${}^\bullet e' \cup \underline{e'}$.

On the other hand, procedure pe_update_existing computes new histories for events e' already present in $\mathcal{E}_N$ and consuming (or reading) conditions generated by e. For each one of them, procedure in pe_update_gen_hist (Algorithm 6) generates all the possible histories that e' currently have in $\mathcal{E}_N$.

Finally, pe_update_new examines the original net N and searches for all the events $e' = \langle M_p, M_c, t\rangle$ that can be appended to $\mathcal{E}_N$ due to the fact that they now have at least one history thanks to the addition of $\langle e, H\rangle$ to $\mathcal{E}_N$. Algorithm 4 generates, thus, enriched events such that event part is new in $\mathcal{E}_N$ (of course, also the history part).

The rationale behind this division division might be understood in the following way. Each new possible extension induced by the addition of $\langle e, H\rangle$ to $\mathcal{E}_N$ is an enriched event $\langle e', H'\rangle$ such that either e' is already present in $\mathcal{E}_N$ (in T') or not. If e' is present

in $\mathcal{E}_N$, it must be one e' for which $e \uparrow e'$ holds. All the cases for such e' are covered by pe_update_context and pe_update_existing. If e' is still not in $\mathcal{E}_N$, it must be one e' from $\mathcal{U}_N$ such that $e \uparrow e'$ holds in $\mathcal{U}_N$. Procedure pe_update_new enumerates all such e' and, with the help of pe_update_gen_hist, filters out those which either are already present in $\mathcal{E}_N$ or do not have a history (in $\mathcal{E}_N$).

Algorithm 2 Procedure pe_update

Require: $\langle e, H\rangle$, the last possible extension appended to $\mathcal{E}_N$
Ensure: The set U contains all the possible extensions induced by $\langle e, H\rangle$
  $U = \mathsf{pe\_update\_context}(e, H)$
  $U = U \cup \mathsf{pe\_update\_existing}(e, H)$
  $U = U \cup \mathsf{pe\_update\_new}(e, H)$
  return U

Algorithm 3 Procedure pe_update_context

Require: $\langle e, H\rangle$, the last possible extension appended to $\mathcal{E}_N$
Ensure: The set U contains all the possible extensions $\langle e', H'\rangle$ induced by $\langle e, H\rangle$ such that $\underline{e} \cap {}^\bullet e' \neq \emptyset$ and $e^\bullet \cap {}^\bullet e' = \emptyset$
  $U = \emptyset$
  for all $e' \in T'$ such that $\underline{e} \cap {}^\bullet e' \neq \emptyset$ and $e^\bullet \cap {}^\bullet e' = \emptyset$ **do**
    for all $H' \in \chi(e')$ **do**
      $f =$ **true**
      for all H'' such that $\langle e', H'\rangle \rightarrow \langle e'', H''\rangle$ is an edge of $\mathcal{H}_N$ and $e \neq e''$ **do**
        $f = f \wedge \neg H \# H''$
        $f = f \wedge H$ does not consume conditions $(e''^\bullet \cup \underline{e''}) \cap ({}^\bullet e' \cup \underline{e'})$
        $f = f \wedge H'$ does not consume conditions $\underline{e} \cap {}^\bullet e'$
      end for
      $\widehat{H} = H \cup H'$
      if f and not $\widehat{H} \in \chi(e')$ **then**
        $U = U \cup \{\langle e', \widehat{H}\rangle\}$
      end if
    end for
  end for
  return U

Algorithm 4 Procedure pe_update_new

Require: $\langle e, H \rangle$, the last possible extension appended to $\mathcal{E}_N$
Ensure: The set U contains all the possible extensions $\langle e', H' \rangle$ induced by $\langle e, H \rangle$ such that e' is still not present in T', and $e^\bullet \cap ({}^\bullet e' \cup \underline{e'}) \neq \emptyset$.

```
U = ∅
for all c ∈ e• do
  Let p = f_P(c)
  for all t such that p ∈ •t ∪ t_ do
    Let {p_1, ..., p_n} = •t
    Let {q_1, ..., q_m} = t_
    for all M_p = {c_1, ..., c_n} such that f_P(c_i) = p_i for 1 ≤ i ≤ n do
      for all M_c = {d_1, ..., d_m} such that f_P(d_i) = q_i for 1 ≤ i ≤ m do
        if not ⟨M_p, M_c, t⟩ ∈ T' then
          U = U ∪ pe_update_gen_hist(⟨M_p, M_c, t⟩)
        end if
      end for
    end for
  end for
end for
return U
```

Algorithm 5 Procedure pe_update_existing

Require: $\langle e, H \rangle$, the last possible extension appended to $\mathcal{E}_N$
Ensure: The set U contains all the possible extensions $\langle e', H' \rangle$ induced by $\langle e, H \rangle$ such that e' is already present in T', and $e^\bullet \cap ({}^\bullet e' \cup \underline{e'}) \neq \emptyset$.

```
U = ∅
for all e' such that e• ∩ (•e' ∪ e'_) ≠ ∅ do
  U = U ∪ pe_update_gen_hist(e')
end for
return U
```

Algorithm 6 Procedure pe_update_gen_hist

Require: An event $e = \langle M_p, M_c, t \rangle$.

Ensure: The set A contains all pairs (e, H) such that $H \in \mathsf{Hist}(e)$ in $\mathcal{U}_N$, and $H \setminus \{e\}$ is a configuration of $\mathcal{E}_N$, and $\langle e, H \rangle$ is not an enriched event of $\mathcal{E}_N$.

```
Let {e_1, ..., e_n} = •M_p ∪ •M_c
for all histories H_1, ..., H_n such that H_1 ∈ χ(e_1) and ... and H_n ∈ χ(e_n) do
  f = true
  for i = 1 to n do
    f = f ∧ H_i does not consume any condition from M_p ∪ M_c
    for j = 1 to i − 1 do
      f = f ∧ ¬H_j#H_i
    end for
  end for
  Ĥ = {e} ∪ H_1 ∪ ... ∪ H_n
  if f and not Ĥ ∈ χ(e) then
    A = A ∪ {(e, Ĥ)}
  end if
end for
return A
```