Efficient Unfolding of Contextual Petri Nets[☆]

Paolo Baldan[a], Alessandro Bruni[a], Andrea Corradini[b], Barbara König[c], César Rodríguez[d], Stefan Schwoon[d]

[a] *Dipartimento di Matematica Pura e Applicata, Università di Padova, Italy*
[b] *Dipartimento di Informatica, Università di Pisa, Italy*
[c] *Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, Germany*
[d] *LSV (ENS Cachan & CNRS) and INRIA, France*

Abstract

A contextual net is a Petri net extended with read arcs, which allows transitions to check for tokens without consuming them. Contextual nets allow for better modelling of concurrent read access than Petri nets, and their unfoldings can be exponentially more compact than those of a corresponding Petri net. A constructive but abstract procedure for generating those unfoldings was proposed in earlier work. However, it remained unclear whether the approach was useful in practice and which data structures and algorithms would be appropriate to implement it. Here, we address this question. We provide two concrete methods for computing contextual unfoldings, with a view to efficiency. We report on experiments carried out on a number of benchmarks. These show that not only are contextual unfoldings more compact than Petri net unfoldings, but they can be computed with the same or better efficiency, in particular with respect to alternative approaches based on encodings of contextual nets into Petri nets.

Keywords: Petri nets, unfolding, asymmetric conflict

1. Introduction

Petri nets are a means for reasoning about concurrent, distributed systems. They explicitly express notions such as concurrency, causality, and independence.

The unfolding of a Petri net is, essentially, an acyclic version of the net in which loops have been unrolled. The unfolding is infinite in general, but for bounded Petri nets one can construct a finite complete prefix of it that completely represents the behaviour of the system, and whose acyclic structure

[☆] Supported by École Doctorale Sciences Pratiques, ENS Cachan, the MIUR project `SisteR`, the University of Padua project `BECOM` and the Regione Toscana project `RUPOS`.
The list of authors is given in alphabetic order.

permits efficient analyses. This prefix is typically much smaller than the reachability graph because an unfolding exploits the inherently concurrent nature of the underlying system; loosely speaking, the more concurrency there is in the net, the more advantages unfoldings have over reachability-graph techniques.

Petri net unfoldings may serve as a basis for further analyses. There is a large body of work describing their construction, their properties, and their use in various fields; see [1] for an extensive survey. For bounded Petri nets, a finite complete prefix of an unfolding can be understood as the compact description of the set of reachable markings of the underlying net. Moreover, while the reachability problem is PSPACE-complete for bounded nets, it is only NP-complete for complete prefixes. Notwithstanding the fact that the size of such a prefix is typically rather larger than the net itself, this opens avenues for efficient reachability checking [2].

However, Petri nets are not well-suited to model concurrent read accesses, that is, multiple actions requiring non-exclusive access to one common resource. The typical way of representing such a situation using a standard net is with "consume-produce loops": each action can consume the common resource when needed, regenerating it immediately after. Unfortunately, this can make the unfolding technique inefficient. In fact, actions reading the common resource are sequentialised so that all their possible interleavings have to be generated, at least in principle. It is possible to mitigate this problem with a place-replication (PR) encoding [3]. Here, a resource with n readers is duplicated n times, and each reader obtains a "private" copy which is accessed with a consume-produce loop. However, the resulting unfolding may still be exponential in n.

Contextual nets explicitly model concurrent read accesses and address this problem. They extend Petri nets with *read arcs*, allowing an action to check for the presence of a resource without consuming it. They have been used, e.g., to model concurrent database access [4], concurrent constraint programs [5], priorities [6], and asynchronous circuits [3]. Their accurate representation of concurrency makes contextual unfoldings up to exponentially smaller in the presence of multiple readers, which promises to yield more efficient analysis procedures.

While the properties and construction of ordinary Petri net unfoldings are well-understood, research on how to construct and exploit the properties of contextual unfoldings has been lacking so far. Contextual unfoldings are introduced in [3, 7], and a first unfolding procedure for a restricted subclass can be found in [3]. A general but non-constructive procedure is proposed in [8].

A constructive, general solution was finally given in [9], at the price of making the underlying theory notably more complicated. In particular, computing a complete prefix required to annotate every event e with a subset of its *histories*; roughly speaking, a history of e is a set of events that must precede e in a possible execution. However, it remained unclear whether the approach could be implemented with reasonable efficiency, and how. For safe nets, the interest of computing a complete contextual prefix was not evident from a practical point of view: while the prefix can be exponentially smaller than the complete prefix of the corresponding PR-encoding, the intermediate structure used to produce it

has asymptotically the same size. More precisely, the number of histories in the contextual prefix matches the number of events in the prefix of the PR-encoding (for general bounded nets, this is not the case).

The purpose of this paper is to address these open issues and to resolve the algorithmic problems related to contextual-net unfolding. In particular, we make the following contributions:

- We provide key elements to implement contextual-net unfoldings efficiently, including data structures and algorithms such as for computing the events of an unfolding (called *possible extensions*) and maintaining a concurrency relation. For the latter, we provide multiple approaches and compare them.
- We generalise the results in [9] in order to deal with (a slight generalisation of) the adequate orders from [10]. Although not very surprising, this extension is quite relevant in practice as it drastically reduces the size of the resulting prefixes.
- We implemented both approaches, aiming for efficiency. The resulting tool, called Cunf [11], matches dedicated Petri net unfolders like Mole [12] on pure Petri nets and additionally handles contextual unfoldings. The new unfolder is not a simple extension of an existing one because the presence of histories influences the data structures at every level.
- We ran the tool on a set of benchmarks and report on the experiments, for both approaches. In particular, it turns out that, even for safe nets, our construction of contextual unfoldings is faster than that for PR-unfoldings.

Apart from details of the prefix computation, our main message is that efficient contextual unfolding is possible and performs better than the PR-encoding, even for safe nets. Contextual nets and their unfoldings therefore have a rightful place in research on concurrency, also from an efficiency point of view.

The paper is structured as follows. Section 2 introduces Petri and contextual nets, and discusses how to encode a contextual net into a Petri net. Sections 3 and 4 recall fundamental notions of contextual unfoldings. Section 5 contains most of the technical results, including two approaches to the unfolding construction. Section 6 discusses data structures and other elements related to the efficient construction of a complete prefix. Section 7 reports on experiments, and Section 8 sketches applications in verification. We conclude in Section 9. Preliminary versions of the contents in this paper have appeared in [13, 14, 15].

2. Basic notions

A *contextual net (c-net)* is a tuple $N = \langle P, T, F, C, m_0 \rangle$, where P and T are disjoint sets of *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 $m_0 \subseteq P$ is the *initial marking*. A pair $(p, t) \in C$ is called *read arc*. The net N is called *finite* when the sets

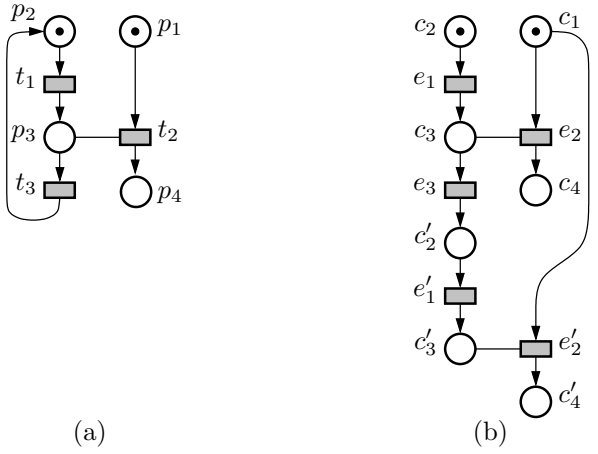

Figure 1: (a) A safe c-net; and (b) an unfolding prefix.

of places and transitions are finite. In general, a *marking* of N is any function $m\colon P \to \mathbb{N}$. The set m_0 is seen as a marking in the obvious way, by letting $m_0(p) = 1$ if $p \in m_0$ and $m_0(p) = 0$, otherwise. A *Petri net* is a c-net without read arcs.

For $x \in P \cup T$, we call ${}^\bullet x := \{\, y \in P \cup T \mid (y,x) \in F \,\}$ the *preset* of x and $x^\bullet := \{\, y \in P \cup T \mid (x,y) \in F \,\}$ the *postset* of x. The *context* of a place p is defined as $\underline{p} := \{\, t \in T \mid (p,t) \in C \,\}$, and the context of a transition t as $\underline{t} := \{\, p \in P \mid (p,t) \in C \,\}$. These notions are extended to sets in the usual fashion. For the sake of simplicity, we assume for any transition t that its context is disjoint from its preset and its postset, i.e. ${}^\bullet t \cap \underline{t} = \emptyset$ and $t^\bullet \cap \underline{t} = \emptyset$.

A set $A \subseteq T$ of transitions is *enabled* at marking m if for all $p \in P$,

$$m(p) \geq |p^\bullet \cap A| + \begin{cases} 1 & \text{if } \underline{p} \cap A \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Such A can *occur* or *be executed*, leading to a new marking m', where $m'(p) = m(p) - |p^\bullet \cap A| + |{}^\bullet p \cap A|$ for all $p \in P$. We call $\langle m, A, m' \rangle$ a *step* of N.[1]

Note that in order to enable concurrently a set of transitions, a marking must include the pre-sets of all transitions and an *additional* token for each place used as context, i.e., a token cannot be read and consumed at the same time. This is needed to ensure that transitions which are concurrently enabled can also fire sequentially in any order. As a consequence concurrency represents event independency, a fact that is at the heart of unfolding approaches. For instance, in the net in Fig. 2, transitions t_1 and t_2 are not concurrently enabled by the

[1] One could define enabledness for multisets of transitions, but this is irrelevant for our purposes. The set-wise definition will be sufficient to illustrate the advantage of c-nets over Petri nets in Section 2.1.

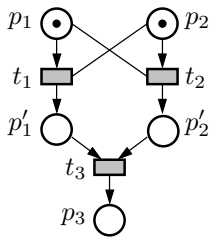

Figure 2: A net with a contextual cycle.

initial marking. Indeed, the firing of t_1 disables t_2 as it consumes a token in p_2, which t_2 must read to fire; similarly, the firing of t_2 disables t_1. Hence t_1 and t_2 cannot occur in the same computation and t_3 is not firable. This dependency is later explained in terms of an asymmetric form of conflict, see Section 4.1. It is worth reminding that different notions of enabling are conceivable (see e.g., [16]) where the simultaneous parallel executions of transitions in contextual cycle, like t_1 and t_2, is allowed.

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 $\langle m_{i-1}, \{t_i\}, m_i \rangle$ is a step for $1 \leq i \leq n$, and m_0 is the initial marking of N; if such a run exists, m_n is said to be *reachable*.

A marking m is *n-safe* if $m(p) \leq n$ for all $p \in P$. A c-net N is said to be n-safe if every reachable marking of N is n-safe. It is called *bounded* if there exists an n such that N is n-safe. A 1-safe net is simply called *safe*. As done for the initial marking, we will occasionally treat 1-safe markings as sets of places. Fig. 1 (a) depicts a safe c-net. Read arcs are drawn as undirected lines. For t_2, we have $\{p_1\} = {}^\bullet t_2$, $\{p_3\} = \underline{t_2}$ and $\{p_4\} = t_2^\bullet$.

Remarks. The class of c-nets used in this paper constrains the initial marking to be a set and does not allow for weights on the flow relation. This restrictions allow for a simplified presentation, in particular because they ensure that conditions and events in the unfolding are identified uniquely by their causal history.

2.1. Encodings of contextual nets

A c-net N can be encoded into a Petri net whose reachable markings are in one-to-one correspondence with those of N. We discuss two such encodings, and illustrate them by the c-net N in Fig. 3 (a). Place p has two transitions b, c in its context, modelling a situation where, e.g., two processes are accessing in a read-only way a common resource represented by p. Note that the step $\{b, c\}$ can occur in N after executing a.

Plain encoding. Given a c-net N, the *plain encoding* of N is the net N' obtained by replacing every read arc (p, t) in the context relation by a consume/produce loop $(p, t), (t, p)$ in the flow relation. The net N' has the same reachable markings as N; it also has the same runs but not the same steps as N. The plain

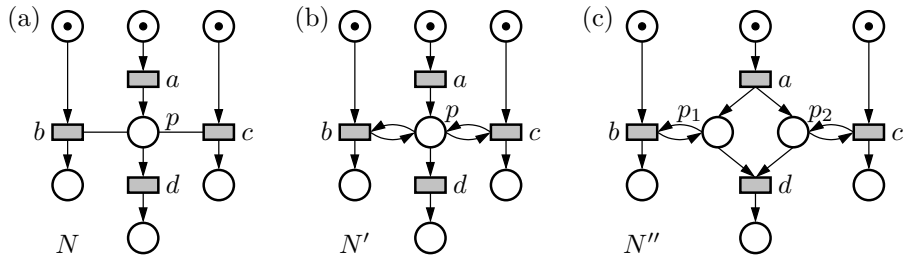

Figure 3: C-net N, its *plain encoding* N' and its *Place-Replication encoding* N''.

encoding of the net N in Fig. 3 (a) is the net N' in Fig. 3 (b). Observe that in N' the firings of $\{b\}$ and $\{c\}$ are sequentialised, hence, after executing a, the step $\{b, c\}$ can no longer occur.

PR-encoding. The *place-replication (PR-) encoding* [3] of a c-net N is a Petri net N'' in which we substitute every place p in the context of $n \geq 1$ transitions $t_1, \ldots, t_n$ by places $p_1, \ldots, p_n$ and update the flow relation of N'' as follows. For all $i \in \{1, \ldots, n\}$,

1. transition t_i consumes and produces place p_i, i.e., $p_i \in {}^\bullet t_i$ and $p_i \in t_i^\bullet$;
2. any transition t producing p in N produces p_i in N'', i.e., $p_i \in t^\bullet$;
3. any transition t consuming p in N consumes p_i in N'', i.e., $p_i \in {}^\bullet t$.

The PR-encoding of the net N in Fig. 3 (a) is the net N'' depicted in Fig. 3 (c). Reachable markings, runs, and steps of N'' are in one-to-one correspondence to those of N.

3. Contextual unfoldings and their prefixes

In this section, we mostly recall basic definitions from [9] concerning unfoldings. We fix a c-net $N = \langle P, T, F, C, m_0 \rangle$ for the rest of the section. Intuitively, the unfolding of N is a safe acyclic c-net where loops of N are "unrolled"; in general, the unfolding is infinite.

Definition 1 (unfolding). *The unfolding of N, denoted by $\mathcal{U}_N$, is a c-net $(B, E, G, D, \widehat{m}_0)$ equipped with a mapping $f\colon (B \cup E) \to (P \cup T)$. We call the elements of B* conditions*, and those of E* events*; f maps conditions to places and events to transitions. We extend f to sets, multisets, and sequences in the usual way; f applied to a marking of $\mathcal{U}_N$ (a set) will yield a marking of N (a multiset).*

Conditions will take the form $\langle p, e' \rangle$, where $p \in P$ and $e' \in E \cup \{\bot\}$, and events will take the form $\langle t, M \rangle$, where $t \in T$ and $M \subseteq B$. We shall assume $f(\langle p, e' \rangle) = p$ and $f(\langle t, M \rangle) = t$, respectively. A set M of conditions is called concurrent*, written $conc(M)$, when $\mathcal{U}_N$ has a reachable marking M' s.t. $M' \supseteq M$.*

Then $\mathcal{U}_N$ is the smallest net containing the following elements:

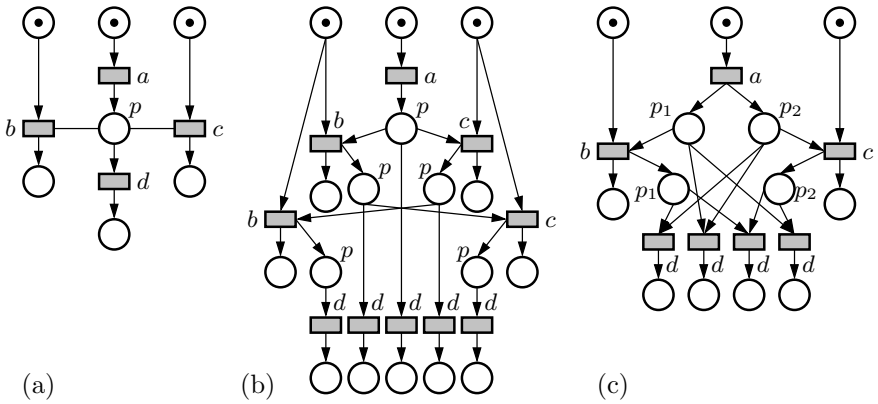

Figure 4: Unfoldings of N, N', and N'' from Fig. 3

- *if* $p \in m_0$*, then* $\langle p, \bot \rangle \in B$ *and* $\langle p, \bot \rangle \in \widehat{m}_0$*;*
- *for any* $t \in T$ *and disjoint pair of sets* $M_1, M_2 \subseteq B$ *such that* $conc(M_1 \cup M_2)$*,* $f(M_1) = {}^\bullet t$*,* $f(M_2) = \underline{t}$*, we have* $e := \langle t, M_1 \cup M_2 \rangle \in E$*, and for all* $p \in t^\bullet$*, we have* $\langle p, e \rangle \in B$*. Moreover,* G *and* D *are such that* ${}^\bullet e = M_1$*,* $\underline{e} = M_2$*, and* $e^\bullet = \{ \langle p, e \rangle \mid p \in t^\bullet \}$*.*

Note that in some previous works on unfoldings the intial marking is omitted from the unfolding because it is determined as their minimal set of places. Here we include the initial marking explicitly to be consistent with the definition of unfoldings as special c-nets.

Fig. 4 shows the unfoldings of the nets from Fig. 3, where f is implicitly indicated by the labels of conditions and events. Note that in this case, the c-net N is isomorphic to its unfolding; crucially, it is smaller than the unfoldings of its two encodings. Call events labelled by b and c "readers", and events labelled by d "consumers". Suppose that we were to replace b and c in Fig. 3 with n transitions reading from p. Then there would be n readers and one consumer in the contextual unfolding; $\mathcal{O}(n!)$ readers *and* consumers in the plain unfolding; and n readers but 2^n consumers in the PR-unfolding.

The net $\mathcal{U}_N$ represents all possible behaviours of N, and in particular a marking m is reachable in N iff some $\widehat{m}$ with $f(\widehat{m}) = m$ is reachable in $\mathcal{U}_N$. Intuitively, the plain unfolding explodes because it represents the step $\{b, c\}$ of the c-net by two runs (in general, by its possible sequentialisations). Instead, in the PR-encoding the consume/produce loops lead to more consuming events in the unfolding.

Definition 2 (causality). *The* causality relation *on* $\mathcal{U}_N$*, denoted* $<$*, is the transitive closure of* $G \cup \{ (e, e') \in E \times E \mid e^\bullet \cap \underline{e'} \neq \emptyset \}$*. For* $x \in B \cup E$*, we write* $[x]$ *for the set of* causes *of* x*, defined as* $\{ e \in E \mid e \leq x \}$*, where* $\leq$ *is the reflexive closure of* $<$*. A set* $X \subseteq E$ *is called* causally closed *if* $[e] \subseteq X$ *for all* $e \in X$*.*

In Fig. 1 (b), we have, e.g., $c_2 < e_1$, $e_1 < e_2$, and $c_2 < e_2$. The causality relation between a pair of events $e < e'$ captures the intuition that e must occur before e' in any run that fires e'.

Definition 3 (prefix). *A* prefix *of $\mathcal{U}_N$ is a net $\mathcal{P} = \langle B', E', G', D', \widehat{m}_0 \rangle$ such that $E' \subseteq E$ is causally closed, $B' = \widehat{m}_0 \cup (E')^\bullet$, and G', D' are the restrictions of G, D to $(B' \cup E')$.*

In other words, a prefix is a causally closed subnet of $\mathcal{U}_N$. Surely, if $\mathcal{P}$ is a prefix and $\widehat{m}$ a marking reachable in it, then $f(\widehat{m})$ is reachable in N. We are interested in computing a prefix for which the inverse also holds.

Definition 4 (finite, complete prefix). *A prefix $\mathcal{P}$ is called* finite *if it contains finitely many events. It is called* complete *if for all markings m of N, m is reachable in N iff there exists a marking $\widehat{m}$ reachable in $\mathcal{P}$ such that $f(\widehat{m}) = m$.*

A finite complete prefix thus preserves all behavioural information about N, while being typically smaller than its reachability graph; its acyclic structure makes the reachability problem easier than for N itself [17]. Moreover, as we saw in Fig. 4, a contextual unfolding is more succinct than its corresponding Petri net unfolding. (In fact, readers familiar with unfoldings may note that in Fig. 4 there exists a complete prefix of the plain unfolding with 2^n rather than $\mathcal{O}(n!)$ reading and consuming events; but this still makes them the largest of the three unfolding types.)

Other papers consider different notions of completeness, requiring e.g. that not only reachable markings, but also the fireability of transitions is preserved (see, e.g., [18]). Roughly, this means that if a cut-off free configuration of the prefix enables a transition, then a representative of that transition should be included in the prefix. This is useful for certain deadlock checking algorithms, and it can be ensured by inserting cut-off events into the prefix. The results presented in the following could be easily adapted accordingly. See our experimental data in Section 7 for another discussion of this issue.

4. Constructing finite complete prefixes

In this section, we study the construction of a finite and complete prefix for a c-net. In Section 4.1, which mostly recalls elements from [9] with minor modifications, we develop a generic algorithm for constructing prefixes. In Section 4.2, we then turn to the question of ensuring that the resulting prefix is complete.

For the rest of the section, we fix a finite c-net N and its unfolding $\mathcal{U}_N$ as in Section 3.

4.1. On finite prefixes

Consider events e_2 and e_3 in Fig. 1 (b). Clearly, $e_2 < e_3$ does not hold. However, any run that fires *both* e_2 and e_3 will fire e_2 before e_3 (since e_3 consumes c_3). This situation arises due to read arcs and motivates the next definition.

Definition 5 (asymmetric conflict). *Two events $e, e' \in E$ are in* asymmetric conflict*, written $e \nearrow e'$, iff (i) $e < e'$, or (ii) $e \cap {}^\bullet e' \neq \emptyset$, or (iii) $e \neq e'$ and ${}^\bullet e \cap {}^\bullet e' \neq \emptyset$. For a set of events $X \subseteq E$, we write $\nearrow_X$ to denote the relation $\nearrow \cap (X \times X)$.*

Note that the asymmetric-conflict relation $\nearrow$ gives rise to a digraph $(E, \nearrow)$, which we henceforth identify with $\nearrow$ itself.

An asymmetric conflict can be thought of as a scheduling constraint: if both e, e' occur in a run, then e must occur first. Note that in case (iii) this is vacuously true, as e, e' cannot both occur. Thus, by condition (iii) $\nearrow$ subsumes the symmetric conflicts known from Petri net unfoldings as cycles of length two.

Definition 6 (configuration). *A* configuration *of the unfolding $\mathcal{U}_N$ is a finite, causally closed set of events $\mathcal{C}$ such that $\nearrow_{\mathcal{C}}$ is acyclic. $Conf(\mathcal{U}_N)$ denotes the set of all such configurations.*

Thus a set of events is a configuration iff all its events can be ordered to form a run that respects the scheduling constraints given by $\nearrow$.

Definition 7 (order and conflict on configurations). *We say that configuration $\mathcal{C}$* evolves *to configuration $\mathcal{C}'$, written $\mathcal{C} \sqsubseteq \mathcal{C}'$, iff $\mathcal{C} \subseteq \mathcal{C}'$ and $\neg(e' \nearrow e)$ for all $e \in \mathcal{C}$ and $e' \in \mathcal{C}' \setminus \mathcal{C}$. By $\mathcal{C} \sqsubset \mathcal{C}'$ we denote $\mathcal{C} \sqsubseteq \mathcal{C}'$ and $\mathcal{C} \neq \mathcal{C}'$.*

Configurations $\mathcal{C}$, $\mathcal{C}'$ are said to be in conflict*, written $\mathcal{C} \# \mathcal{C}'$, when there is no configuration $\mathcal{C}''$ satisfying $\mathcal{C} \sqsubseteq \mathcal{C}''$ and $\mathcal{C}' \sqsubseteq \mathcal{C}''$.*

Note that $\sqsubseteq$ is not merely the subset relation between configurations. Intuitively, $\mathcal{C} \sqsubseteq \mathcal{C}'$ when a run of $\mathcal{C}$ can be extended into a run of $\mathcal{C}'$. Instead, $\mathcal{C} \# \mathcal{C}'$ when they cannot evolve to a common future configuration. For instance, in Fig. 1 (b) we have $\{e_1, e_3\} \not\sqsubseteq \{e_1, e_2, e_3\}$ (and actually $\{e_1, e_3\} \# \{e_1, e_2, e_3\}$) because e_2 would have to fire before e_3. However, if two configurations are *not* in conflict, then their union is a configuration.

Remark 1. *[9] For two configurations $\mathcal{C}_1, \mathcal{C}_2$, we have $\mathcal{C}_1 \# \mathcal{C}_2$ iff there exists $e_1 \in \mathcal{C}_1$ and $e_2 \in \mathcal{C}_2 \setminus \mathcal{C}_1$ such that $e_2 \nearrow e_1$, or the symmetric condition holds.*

The *cut* of a configuration $\mathcal{C}$ is the marking reached in $\mathcal{U}_N$ by a run of $\mathcal{C}$. We define $\mathsf{Cut}(\mathcal{C}) := (\widehat{m}_0 \cup \mathcal{C}^\bullet) \setminus {}^\bullet\mathcal{C}$. The *marking* of $\mathcal{C}$ is its image through f: $\mathsf{Mark}(\mathcal{C}) := f(\mathsf{Cut}(\mathcal{C}))$.

Definition 8 (history). *Let e be an event and $\mathcal{C}$ a configuration with $e \in \mathcal{C}$. We call the configuration $\mathcal{C}[\![e]\!] := \{\, e' \in \mathcal{C} \mid e'(\nearrow_{\mathcal{C}})^* e \,\}$ the* history of e in $\mathcal{C}$*. Moreover, $Hist(e) := \{\, \mathcal{C}[\![e]\!] \mid \mathcal{C} \in Conf(\mathcal{U}_N) \wedge e \in \mathcal{C} \,\}$ is the set of histories of e.*

Remark 2. *Let $\mathcal{C}$ be a configuration, and $e \in \mathcal{C}$. Then $\mathcal{C}[\![e]\!] \sqsubseteq \mathcal{C}$.*

While in Petri net unfoldings each event has exactly one history, a contextual unfolding may have multiple (even infinitely many) histories per event. For instance, in Fig. 1 (b) $Hist(e_3) = \{\{e_1, e_3\}, \{e_1, e_2, e_3\}\}$. To compute a complete prefix, one annotates events with a finite subset of their histories.

Definition 9 (enriched prefix). *An* enriched event *is a pair* $\langle e, H\rangle$ *where* $e \in E$ *and* $H \in \mathit{Hist}(e)$. *An* enriched prefix (EP) *of* $\mathcal{U}_N$ *is a pair* $\mathcal{E} = \langle \mathcal{P}, \chi\rangle$ *such that* $\mathcal{P} = \langle B', E', G', D', \widehat{m}_0\rangle$ *is a prefix and* $\chi\colon E' \to 2^{2^E}$ *satisfies for all* $e \in E'$ *(i)* $\emptyset \neq \chi(e) \subseteq \mathit{Hist}(e)$, *and (ii)* $H \in \chi(e)$ *and* $e' \in H$ *imply* $H[\![e']\!] \in \chi(e')$. *For an enriched event* $\langle e, H\rangle$, *we write* $\langle e, H\rangle \in \mathcal{E}$ *if* $e \in E'$ *and* $H \in \chi(e)$.

In [9], a complete prefix of $\mathcal{U}_N$ is constructed by a saturation procedure that adds one enriched event at a time until there remains no addition that would "contribute" new markings. We make this idea concrete in the following:

Definition 10 (possible extension). *An enriched event* $\langle e, H\rangle$ *is a* possible extension *of an EP* $\mathcal{E}$ *if* $\langle e', H[\![e']\!]\rangle \in \mathcal{E}$ *for all* $e' \in H \setminus \{e\}$ *and* $\langle e, H\rangle \notin \mathcal{E}$.

The algorithm will take into account possible extensions in some suitable order. Let $\prec$ be an order on configurations. If $\prec$ satisfies that $\mathcal{C} \prec \mathcal{C}'$ for any $\mathcal{C}, \mathcal{C}'$ such that $\mathcal{C} \sqsubset \mathcal{C}'$, then we call $\prec$ *basic*. We extend $\prec$ to enriched events by $\langle e, H\rangle \prec \langle e', H'\rangle$ if $H \prec H'$. For a fixed $\prec$, a tuple $\langle e, H\rangle$ is called *cutoff* iff there exists an enriched event $\langle e', H'\rangle$ such that $\mathsf{Mark}(H') = \mathsf{Mark}(H)$ and $\langle e', H'\rangle \prec \langle e, H\rangle$. Any basic order $\prec$ parametrises the following informal algorithm for constructing an EP of $\mathcal{U}_N$.

Algorithm 1.

- Start with the EP that contains just $\widehat{m}_0$;
- Then, in each iteration, add a non-cutoff $\prec$-minimal possible extension.
- If no non-cutoff possible extensions remain, terminate.

Remark 3. *Notice that Algorithm 1 maintains condition (ii) of Definition 9, due to the choice of possible extensions in Definition 10: condition (ii) is clearly satisfied initially, where there are no events; and every addition of a possible extension maintains this invariant. The EP is thus* closed *in the sense of [9], Definition 13.*

Algorithm 1 terminates if N is bounded. The size and shape of the produced prefix depends on the choice of $\prec$. In particular, the resulting prefix may be complete or not. In Section 4.2, we discuss how to choose $\prec$ so that the prefix is guaranteed to be both complete and not larger than the size of the reachability graph. Later, in Section 5 and Section 6 we will discuss how to implement Algorithm 1 efficiently.

4.2. On complete prefixes

Algorithm 1 was first introduced in [9], where it was shown that the resulting prefix is complete when $\prec$ is following partial order:

$$\mathcal{C} \prec \mathcal{C}' \quad \text{iff} \quad |\mathcal{C}| < |\mathcal{C}'| \tag{1}$$

This condition was originally introduced by McMillan [17] in his seminal paper on Petri net unfoldings. However, it is known that McMillan's order may create complete prefixes that are up to exponentially larger than the reachability graph [10]. This is because $\prec$ is a partial order: multiple enriched events may lead to the same marking, but if they are incomparable (because their histories have the same size), then none of them is a cutoff. It is therefore preferable to replace McMillan's order by a suitable finer order, ideally a *total* order, in which case the resulting prefix will have at most as many events as there are reachable markings in the net (and usually far fewer).

For Petri nets, this problem was resolved in [10], which introduces *adequate* orders. Any adequate order will yield a complete prefix, and [10] exhibits an adequate order that is total for safe nets. Below, in Definition 11, we present a slight generalisation of the adequate orders from [10] that is more suitable for c-nets. We then show that these orders also yield complete prefixes.

We first adapt the notion of extension for a configuration. Given a configuration $\mathcal{C}$, one calls a set of events $\mathcal{X}$ an *extension* of $\mathcal{C}$ if $\mathcal{C} \cap \mathcal{X} = \emptyset$ and $\mathcal{C} \cup \mathcal{X}$ is a configuration such that $\mathcal{C} \sqsubseteq \mathcal{C} \cup \mathcal{X}$. We call two extensions $\mathcal{X}_1, \mathcal{X}_2$ (of different configurations) *isomorphic* if the subnets labelled by f and consisting of the events $\mathcal{X}_1$, conditions ${}^\bullet\mathcal{X}_1 \cup \mathcal{X}_1^\bullet \cup \underline{\mathcal{X}}_1$ (resp. $\mathcal{X}_2$) and G, D restricted to these events and conditions are isomorphic. It is evident that two configurations with the same associated markings have the same extensions, modulo isomorphism.

Definition 11 (adequate order). *Given a partial order $\prec$ on configurations, we call it* adequate *iff it satisfies the following properties:*

1. *$\prec$ is well founded;*
2. *$\mathcal{C}_1 \sqsubset \mathcal{C}_2$ implies $\mathcal{C}_1 \prec \mathcal{C}_2$;*
3. *$\prec$ is preserved by finite extensions, that is, if $\mathcal{C}_1 \prec \mathcal{C}_2$, and $\mathsf{Mark}(\mathcal{C}_1) = \mathsf{Mark}(\mathcal{C}_2)$, for any extension $\mathcal{X}$ of $\mathcal{C}_1$ there exists some extension $\mathcal{X}'$ of $\mathcal{C}_2$ isomorphic to $\mathcal{X}$ such that $\mathcal{C}_1 \cup \mathcal{X} \prec \mathcal{C}_2 \cup \mathcal{X}'$.*

The above notion of adequate order differs from the one in [10] for condition 2, which is $\mathcal{C}_1 \subset \mathcal{C}_2$ there. Note that for Petri net unfoldings $\mathcal{C}_1 \subset \mathcal{C}_2$ implies $\mathcal{C}_1 \sqsubset \mathcal{C}_2$, hence the two notions coincide. However, for contextual unfoldings this is not the case; for instance, in Fig. 1 $\{e_1, e_3\} \not\sqsubset \{e_1, e_2, e_3\}$. For c-nets therefore, Definition 11 is a slight generalisation of [10].

Proposition 1 (completeness of the prefix). *Let N be a finite bounded c-net. If $\prec$ is adequate, then Algorithm 1 terminates with an EP $\mathcal{E} = \langle \mathcal{P}, \chi \rangle$ such that $\mathcal{P}$ is a complete prefix of $\mathcal{U}_N$.*

Proof The proof consists of two parts: first one shows that the algorithm terminates, then one shows that the result is complete. The structure of the proof mimics the proof from [10], except that it has to be lifted to enriched events.

To prove that the algorithm terminates, one exploits that N is bounded, therefore the number of reachable markings is bounded; this ensures that along

any infinite chain $\mathcal{C}_1 \sqsubset \mathcal{C}_2 \sqsubset \cdots$ there are two configurations $\mathcal{C}_1, \mathcal{C}_2$ with $\mathsf{Mark}(\mathcal{C}_1) = \mathsf{Mark}(\mathcal{C}_2)$ and hence a cutoff.

To show that the resulting prefix is complete, let us say that $\mathcal{E}$ contains a configuration $\mathcal{C}$ of $\mathcal{U}_N$ if all events of $\mathcal{C}$ are in $\mathcal{E}$ and $\langle e, \mathcal{C}[\![e]\!]\rangle \in \mathcal{E}$ for all $e \in \mathcal{C}$. Let m be a reachable marking in N. Then there exists a configuration $\mathcal{C}$ of $\mathcal{U}_N$ such that $\mathsf{Mark}(\mathcal{C}) = m$. Either $\mathcal{C}$ is contained in $\mathcal{E}$ (and we are done), or C contains an enriched cutoff event $\langle e, \mathcal{C}[\![e]\!]\rangle$. In the latter case, there exists a $\prec$-smaller enriched event $\langle e', \mathcal{C}'\rangle \in \mathcal{U}_N$ with $\mathsf{Mark}(\langle e', \mathcal{C}'\rangle) = \mathsf{Mark}(\langle e, \mathcal{C}[\![e]\!]\rangle)$. We can then construct isomorphic extensions $\mathcal{X}, \mathcal{X}'$ of $\mathcal{C}[\![e]\!]$ and $\mathcal{C}'$ and thus obtain (thanks to condition 3 of Definition 11) a $\prec$-smaller configuration with marking m. Since $\prec$ is well-founded, this argument can be iterated only finitely many times, thus resulting eventually in a configuration contained in $\mathcal{E}$. □

5. Two approaches to possible extensions and concurrency

We now turn to the question of how to implement Algorithm 1 efficiently, for constructing unfoldings in practice. Notice that Algorithm 1 is parametrised by an ordering $\prec$ on enriched events. While that order needs to be adequate to obtain complete prefixes (see Section 4.2), the results in this section require only that $\prec$ be basic.

Let N and $\mathcal{U}_N$ be, as in the previous sections, a fixed finite c-net and its unfolding. The main computational problem of Algorithm 1 is to identify the possible extensions in each iteration. For Petri net unfolders (which do not deal with histories) this involves identifying sets M of conditions such that $conc(M)$ and $f(M) = {}^\bullet t$ for some $t \in T$ (compare Definition 1). For Petri nets, it is known that $conc(M)$ holds iff $conc(\{c_1, c_2\})$ for all pairs $c_1, c_2 \in M$. Possible extensions can therefore be identified by repeatedly consulting a *binary* relation on conditions. Moreover, this binary relation can be computed efficiently and incrementally during prefix construction. This idea is exploited by existing tools such as Mole [12] or Punf [19].

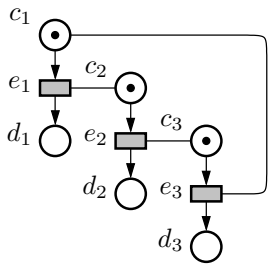

Figure 5: A net showing that concurrency is not a binary relation for c-net unfoldings.

The above statement about $conc(\cdot)$ is invalid for contextual unfoldings. Consider the net in Figure 5, which is identical to its unfolding, and the set

$M = \{d_1, d_2, d_3\}$. Clearly, the elements of M are pairwise concurrent, for instance d_1, d_2 may be covered by firing e_1, then e_2. However, $conc(M)$ does not hold. In fact, $[\{d_1, d_2, d_3\}] = \{e_1, e_2, e_3\}$ cannot be fired in the same run as $e_1 \nearrow e_2 \nearrow e_3 \nearrow e_1$, i.e., it includes an asymmetric-conflict cycle and thus it is not a configuration.

In the following, we introduce a binary relation for c-net unfoldings in which pairwise concurrency does imply reachability of the whole set. This relation is defined on conditions enriched with histories.

Definition 12 (histories for conditions). *Let c be a condition. A* generating history *of c is $\emptyset$ if $c \in \widehat{m}_0$, or $H \in Hist(e)$, where $\{e\} = {}^\bullet c$. A* reading history *of c is any $H \in Hist(e)$ such that $e \in \underline{c}$. A* history *of c is any of its generating or reading histories or $H_1 \cup H_2$, where H_1 and H_2 are histories of c verifying $\neg(H_1 \# H_2)$. In the latter case, the history is called* compound.

In words, for a condition c, not belonging to the initial marking, a generating history is any history of the unique event producing of c. When c is in the initial marking it has only an empty generating history. A reading history is any history of the events reading c. Compound histories are combinations of generating and (possibly multiple) reading histories.

If H is a history of c, we call $\langle c, H\rangle$ an *enriched condition*, referred to as generating, reading, or compound condition, according to H. For an EP $\mathcal{E} = \langle \mathcal{P}, \chi\rangle$, we say $\langle c, H\rangle \in \mathcal{E}$ if for all $e \in {}^\bullet c \cup \underline{c}$ it holds $C[\![e]\!] \in \chi(e)$, i.e., H is built from histories in χ. The mapping f is extended to enriched events and conditions by $f(\langle e, H\rangle) = f(e)$ and $f(\langle c, H\rangle) = f(c)$.

Definition 13 (concurrency for enriched conditions). *Two enriched conditions $\langle c, H\rangle, \langle c', H'\rangle$ are called* concurrent*, written $\langle c, H\rangle \parallel \langle c', H'\rangle$, iff $\neg(H \# H')$ and $c, c' \in \mathsf{Cut}(H \cup H')$.*

To illustrate the definition, we give some examples from Fig. 5.

- $\langle c_1, \emptyset\rangle \not\parallel \langle d_1, \{e_1\}\rangle$ because $c_1 \notin \mathsf{Cut}(\{e_1\})$;
- $\langle d_1, \{e_1\}\rangle \not\parallel \langle d_2, \{e_2\}\rangle$ because $\{e_1\} \# \{e_2\}$;
- $\langle c_1, \emptyset\rangle \parallel \langle d_3, \{e_3\}\rangle$;
- $\langle d_1, \{e_1\}\rangle \parallel \langle d_2, \{e_1, e_2\}\rangle$.

Remark 4. *Given enriched conditions $\rho = \langle c, H\rangle$ and $\rho' = \langle c', H'\rangle$ the statement $\rho \parallel \rho'$ 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. $\neg(\exists e \in H,\, c' \in {}^\bullet e)$
4. $\neg(\exists e \in H',\, c \in {}^\bullet e)$

The following facts will be useful in the subsequent proofs.

Remark 5. *Let H, H' be two histories of the same event e. Then $\neg(H \# H')$ if and only if $H = H'$. In fact, $\neg(H \# H')$ iff there exists a configuration $\mathcal{C}$ such that $H, H' \sqsubseteq \mathcal{C}$ iff $H = C[\![e]\!] = H'$.*

Similarly, if $\rho = \langle c, H\rangle$ and $\rho' = \langle c, H'\rangle$ are two generating conditions for the same c, then $\rho \parallel \rho'$ if and only if $H = H'$. In fact, $\neg(H \# H')$ if and only if $H = H'$. To see this, note that either $c \in \widehat{m}_0$, then $H = H' = \emptyset$. Or ${}^\bullet c = \{e\}$, in which case the result follows from the above observation.

We introduce another notion, that of an ancestor:

Definition 14 (ancestor). *Let $\rho = \langle c, H\rangle$ be a reading history. The* ancestor *of ρ, denoted $\rho^\uparrow$, is the unique generating condition $\langle c, H'\rangle$ such that $H' \sqsubseteq H$.*

Note that indeed H' is uniquely determined due to Remark 5.

In Section 5.1, we discuss how relation $\parallel$ helps to compute possible extensions. For this purpose, we propose two methods that we call *lazy* and *eager*. In Section 5.2 we then discuss how to update $\parallel$ during the unfolding construction. In Section 5.3 we show how to obtain a unique decomposition for each possible extension, and in Section 5.4 we compare the lazy and the eager approaches from a theoretical point of view.

5.1. Computing possible extensions

We discuss two ways of computing possible extensions. The first, called "lazy", avoids constructing compound conditions (see Definition 12), reducing the number of enriched conditions considered. The second, "eager" approach does use compound conditions, saving work when computing possible extensions instead.

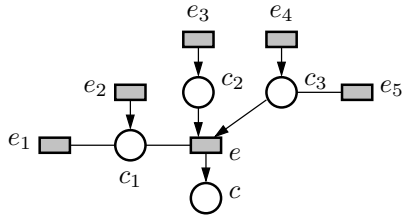

Figure 6: Predecessors w.r.t. asymmetric conflict of an event e.

Lazy Approach. The lazy approach is based on the observation that the history associated with an event can be constructed by taking generating and reading histories for places in the pre-set and generating histories for places in the context. This is stated by the following proposition:

Proposition 2 (possible extensions - lazy). *The pair $\langle e, H\rangle$ with $f(e) = t$ is an enriched event iff there exist sets X_p, X_c of enriched conditions such that*

1. *$f(X_p) = {}^\bullet t$ and $f(X_c) = \underline{t}$;*

2. X_p *contains generating or reading conditions,* X_c *generating conditions;*
3. $X_p \cup X_c$ *contains exactly one generating condition for every* $c \in ({}^\bullet e \cup \underline{e})$*;*
4. *for all* $\rho, \rho' \in X_p \cup X_c$ *we have* $\rho \parallel \rho'$*;*
5. *finally,* $H = \{e\} \cup \bigcup_{\langle c, H' \rangle \in X_p \cup X_c} H'$.

Proposition 2 allows to identify new possible extensions whenever a prefix is extended with new enriched conditions. Compound conditions are avoided at the price of allowing X_p to contain, for every $c \in {}^\bullet e$, an arbitrary number of reading conditions.

To illustrate the meaning of X_p and X_c in Proposition 2, consider Fig. 6. To create a history for event e, X_p must contain generating histories for c_2 and c_3, i.e. histories of events e_3 and e_4. Optionally, X_p may contain a reading history of c_3 coming from e_5. As for X_c, it must contain a generating history of c_1 (coming from e_2) but it must not contain a reading history of c_1. In fact, note that e_1 is not an asymmetric-conflict predecessor of e, hence it is not included in any history of e.

Proof

Left-to-right.

Assume that $\langle e, H \rangle$ is an enriched event. We define $X_p := X_1 \cup X_2$, where X_1 (resp. X_2) are sets of generating (resp. reading) conditions obtained from the generating (resp. reading) histories of ${}^\bullet e$ that extend to H:

$$\begin{aligned} X_1 &= \{ \langle c, \emptyset \rangle \mid c \in {}^\bullet e \cap \widehat{m}_0 \} \cup \{ \langle c, H[\![e']\!] \rangle \mid c \in {}^\bullet e \setminus \widehat{m}_0 \wedge \{e'\} = {}^\bullet c \} \\ X_2 &= \{ \langle c, H[\![e']\!] \rangle \mid c \in {}^\bullet e \wedge e' \in \underline{c} \cap H \} \end{aligned}$$

For X_c, we take the generating histories of conditions in $\underline{e}$:

$$X_c = \{ \langle c, \emptyset \rangle \mid c \in \underline{e} \cap \widehat{m}_0 \} \cup \{ \langle c, H[\![e']\!] \rangle \mid c \in \underline{e} \setminus \widehat{m}_0 \wedge \{e'\} = {}^\bullet c \}$$

This choice of X_p and X_c evidently satisfies properties 1, 2, and 3 of Proposition 2. As for the rest:

Property 4. Let $\langle c_1, H_1 \rangle, \langle c_2, H_2 \rangle \in X_p \cup X_c$. Then $H_1 \sqsubseteq H$ and $H_2 \sqsubseteq H$, so $\neg(H_1 \# H_2)$. Moreover, $c_1 \in \mathsf{Cut}(H_1)$ and $c_2 \in \mathsf{Cut}(H_2)$. Assume w.l.o.g. that $c_1 \notin \mathsf{Cut}(H_1 \cup H_2)$. Then there exists $e' \in H_2$ such that $c_1 \in {}^\bullet e'$. Since $c_1 \in {}^\bullet e \cup \underline{e}$, we have $e \nearrow e'$. Moreover, $e' \in H_2 \sqsubseteq H$, where the configuration H is a history of e, therefore by definition $e' \nearrow_H^* e$. This is a contradiction because H may not contain asymmetric conflict cycles.

Property 5. Recall that any $e' \in H$ satisfies $e' \nearrow_H^* e$. So either $e' = e$ or there exists $e'' \in H$ such that $e' \nearrow_H^* e''$ and $e'' \nearrow e$. From all such e'', pick one that is maximal w.r.t. $<$. According to Definition 2 and Definition 5, this leaves three cases: there is c such that either (i) $c \in {}^\bullet e$ and $e'' \in {}^\bullet c$, or (ii) $c \in {}^\bullet e$ and $e'' \in \underline{c}$, or (iii) $c \in \underline{e}$ and $e'' \in {}^\bullet c$. Moreover, since $e'' \in H$, we conclude that $H'' := H[\![e'']\!]$ is a history such that $H'' \sqsubseteq H$. Thus, in cases (i) and (ii), $\langle c, H'' \rangle \in X_p$, and in case (iii), $\langle c, H'' \rangle \in X_c$. Finally, $e' \in H''$ because $e' \in H$ and $e' \nearrow_H^* e''$.

Right-to-left.

Suppose that there exist X_p, X_c, and H fulfilling properties 1–5. We have to show that H is a history of e. To see this, it suffices to see that H is a configuration and that $H[\![e]\!]$ equals H.

H is causally closed. Any H' such that $\langle e', H' \rangle \in X_p \cup X_c$ is causally closed. Moreover, the choice of X_p ensures that all causal predecessors of e are contained in H.

$\nearrow_H$ is acyclic. By contradiction, assume that there exists a simple cycle $e_1 \nearrow_H e_2 \nearrow_H \cdots \nearrow_H e_n \nearrow_H e_1$, for some $n \geq 2$.

Either e appears in the cycle, then w.l.o.g. $e_1 = e$ and $e_2 \in H_2$, for some $\langle c_2, H_2 \rangle \in X_p \cup X_c$. Now, $e \nearrow e_2$ implies (see Definition 5) either $e < e_2$, $\underline{e} \cap {}^\bullet e_2 \neq \emptyset$, or ${}^\bullet e \cap {}^\bullet e_2 \neq \emptyset$. In all cases, H_2 consumes a token from some $c_1 \in {}^\bullet e \cup \underline{e}$. Due to property 3, $X_p \cup X_c$ contains some tuple $\langle c_1, H_1 \rangle$, but $\langle c_1, H_1 \rangle \not\parallel \langle c_2, H_2 \rangle$, violating property 4.

Or e does not appear in the cycle, then w.l.o.g. $e_2 \in H_2$ and $e_1 \in H_1 \setminus H_2$ for some $\langle c_1, H_1 \rangle, \langle c_2, H_2 \rangle \in X_p \cup X_c$, where $H_1 \neq H_2$. Thus, $H_1 \# H_2$, violating property 4.

$H[\![e]\!] = H$. We need to show that $e' \nearrow_H^* e$ holds for each $e' \in H$. Recall that the elements $\langle c', H' \rangle \in X_p \cup X_c$ are chosen such that H' is either empty or a history of some e'' such that $e'' \nearrow e$. Thus, if $e' \neq e$, we have $e' \nearrow_H^* e'' \nearrow e$ for some suitable e'', and for $e' = e$ the condition holds trivially. □

Eager approach. The eager approach, instead of attempting to combine generating and reading histories when computing a possible extension, explicitly produces all types of enriched conditions, including compound ones. This means more enriched conditions, but on the other hand less work when computing possible extensions.

Proposition 3 (possible extensions - eager). *The pair $\langle e, H \rangle$ with $f(e) = t$ is an enriched event iff there exist sets X_p, X_c of enriched conditions such that*

1. *$f(X_p) = {}^\bullet t$ and $f(X_c) = \underline{t}$;*
2. *X_p contains arbitrary enriched conditions, X_c generating conditions;*
3. *$X_p \cup X_c$ contains exactly one enriched condition for every $c \in ({}^\bullet e \cup \underline{e})$;*
4. *for all $\rho, \rho' \in X_p \cup X_c$ we have $\rho \parallel \rho'$;*
5. *finally, $H = \{e\} \cup \bigcup_{\langle c, H' \rangle \in X_p \cup X_c} H'$.*

Before we proceed with the proof, let us make some remarks. First, notice that $|X_p| = |{}^\bullet t|$ by properties 1 and 3 in Proposition 3 whereas no such bound exists in Proposition 2. Like the latter, Proposition 3 allows to identify new possible extensions upon addition of new enriched conditions.

As an example, consider again Fig. 6. The set X_p must contain an arbitrary history for c_2 and c_3. Concretely, for c_2 we can take only a generating history (coming from e_3), while for c_3 we can use a generating history (coming from e_4) or a compound history (combining histories of e_4 and e_5). Instead, X_c is still restricted to include generating histories only, in this case of c_1.

Second, one can establish a relation between possible extensions according to Proposition 2 and Proposition 3. Let X_p be a set as in Proposition 2. We define

$$\mathsf{Eager}(X_p) := \{\, \langle c, \bigcup_{\langle c,H\rangle \in X_p} H\rangle \mid c \in {}^\bullet e \,\}.$$

On the other hand, if X_p is a set as in Proposition 3, let

$$\mathsf{Lazy}(X_p) := \{\, \langle c, H[\![e']\!]\rangle \mid \langle c, H\rangle \in X_p \wedge e' \in ({}^\bullet c \cup \underline{c}) \cap H \,\} \cup \{\, \langle c, \emptyset\rangle \mid c \in {}^\bullet e \cap \widehat{m}_0 \,\}.$$

We are now ready to state the proof of Proposition 3.

Proof (of Proposition 3) We shall prove that a collection of enriched conditions satisfying properties 1–5 in Proposition 2 exists if and only if such a collection exists for properties 1–5 in Proposition 3.

Left-to-right.

Let X_p, X_c be a pair of sets as per Proposition 2. We define $X'_p := \mathsf{Eager}(X_p)$ and prove that X'_p, X_c satisfy properties 1–5 of Proposition 3.

Properties 1 and 5 are immediate. For properties 2 and 3 it suffices to realise that the elements of X_p are concurrent, therefore their union forms a valid compound history according to Definition 12.

For property 4, let $\rho = \langle c, H_1 \cup \cdots \cup H_m\rangle, \rho' = \langle c', H_{m+1}, \cup \cdots \cup H_n\rangle \in X'_p \cup X_c$ such that $\langle c, H_i\rangle, \langle c', H_j\rangle \in X_p \cup X_c$ for $1 \leq i \leq m$ and $m < j \leq n$. The elements of $X_p \cup X_c$ are concurrent, so $c, c' \in \mathsf{Cut}(H_1 \cup \cdots \cup H_n)$. Moreover, if $(H_1 \cup \cdots \cup H_m) \# (H_{m+1} \cup \cdots \cup H_n)$, then $H_i \# H_j$ for some $1 \leq i \leq m < j \leq n$, which contradicts the assumption that $X_p \cup X_c$ are pairwise concurrent. Thus, $\rho \parallel \rho'$ holds.

Right-to-left.

Let X_p, X_c be a pair of sets as per Proposition 3. We define $X'_p := \mathsf{Lazy}(X_p)$. Now we show that X'_p, X_c satisfy properties 1–5 of Proposition 2.

Properties 1, 2, and 3 are immediate from the definition of X'_p, X_c. Property 4 follows from pairwise concurrency in X_p, X_c. For property 5, we need to show that every event $e' \in H$ is included in one of the elements of $X'_p \cup X_c$. For $e' = e$, this is immediate. Otherwise there exists a chain $e' \nearrow_H \cdots \nearrow_H e'' \nearrow_H e$ such that $e'' \in {}^\bullet({}^\bullet e) \cup \underline{{}^\bullet e} \cup {}^\bullet(\underline{e})$. The definition of X'_p and X_c implies that their union contains at least one tuple $\langle c, H[\![e'']\!]\rangle$, where $e' \in H[\![e'']\!]$. □

5.2. Updating the concurrency relation

Propositions 2 and 3 tell us how to identify possible extensions: it suffices to identify a set of concurrent enriched conditions satisfying suitable side conditions. We thus need a technique for efficiently computing the binary concurrency

relation $\|$ on enriched conditions. In the following, we discuss methods that allow to do this incrementally, i.e., by extending $\|$ whenever the unfolding grows by the insertion of new enriched events.

Again, it shall be useful to contrast our approach with that for Petri nets. There, when an event e is created with concurrent preset M, a condition c in $e^\bullet$ is concurrent with other conditions in $e^\bullet$, non-concurrent with the elements of M, and for any other condition c' we have $conc(\{c, c'\})$ iff $conc(\{c_i, c'\})$ for all $c_i \in M$.

This principle is not correct for c-nets, even when lifted to enriched conditions. Consider Fig. 1 (b). Let us consider the enriched conditions $\rho_1 = \langle c_3, \{e_1\}\rangle$, $\rho = \langle c'_2, \{e_1, e_3\}\rangle$, and $\rho' = \langle c_4, \{e_1, e_2\}\rangle$. Now, ρ_1 is used to create the possible extension $\langle e_3, \{e_1, e_3\}\rangle$, which gives rise to the enriched condition ρ. Observe that $\rho_1 \parallel \rho'$ holds but $\rho \parallel \rho'$ does not. Intuitively, this is because ρ' contains an event (e_2) that reads, without consuming, a condition (c_3) in ${}^\bullet e_3$, and such event is not included in ρ_1. For computing the concurrency relation $\|$, we must therefore introduce an additional condition ensuring that such events are taken into account correctly.

The following two results show how to achieve this. Proposition 4 deals with generating and reading conditions, and Proposition 5 with compound conditions.

Proposition 4 (updating concurrency). *In Algorithm 1, let $\mathcal{E}$ be the current EP, where $\langle e, H\rangle$ is the last addition thanks to sets X_p, X_c as per Proposition 2 or Proposition 3. We denote by $Y_p = e^\bullet \times \{H\}$ and $Y_c = \underline{e} \times \{H\}$ the generating and reading conditions created by the addition of $\langle e, H\rangle$. Let $\rho = \langle c, H\rangle \in Y_p \cup Y_c$, and let $\rho' = \langle c', H'\rangle \in \mathcal{E}$ be any other enriched condition. Then $\rho \parallel \rho'$ iff*

$$\rho' \in Y_p \cup Y_c \ \vee \ (c' \notin {}^\bullet e \ \wedge \ {}^\bullet\underline{e} \cap H' \subseteq H \ \wedge \ \forall \rho_i \in X_p \cup X_c : (\rho_i \parallel \rho'))$$

Proof

Left-to-right.

Assume $\rho \parallel \rho'$ and $\rho' \notin Y_p \cup Y_c$. Since $\neg(H \# H')$, there exists a configuration $\mathcal{C}$ such that $H \sqsubseteq \mathcal{C}$ and $H' \sqsubseteq \mathcal{C}$.

1. Clearly, $c' \notin {}^\bullet e$ is implied by $c' \in \mathsf{Cut}(H \cup H')$.
2. Let e' be an event in ${}^\bullet\underline{e} \cap H'$. Then $e' \nearrow e$. Since $\neg(H \# H')$, and due to Remark 1, e' cannot be in $H' \setminus H$, so $e' \in H$.
3. Let $\rho_i = \langle c_i, H_i\rangle \in X_p \cup X_c$. Then $H_i \sqsubseteq H \sqsubseteq \mathcal{C}$, therefore $\neg(H_i \# H')$. Moreover $c' \in \mathsf{Cut}(H_i \cup H')$. In fact, otherwise there would exist $e_1 \in H_i$ that consumes c' and this would contradict $c' \in \mathsf{Cut}(H \cup H')$.
   It remains to show that $c_i \in \mathsf{Cut}(H_i \cup H')$. Assume that there is $e'' \in H'$ such that $c_i \in {}^\bullet e''$. Since $e \nearrow e'' \nearrow^* e'$ and $\neg(H \# H')$, necessarily $e \in H'$, $e \neq e'$, and $H = H'[\![e]\!] \sqsubseteq H'$. Due to condition (ii) of Definition 9 (cf. also Remark 3) $\langle e', H'\rangle \in \mathcal{E}$ implies $\langle e, H\rangle \in \mathcal{E}$. But then, $\langle e, H\rangle$ cannot be a possible extension.

Right-to-left.

We shall show that if the right-hand side of Proposition 4 holds, then $\rho \parallel \rho'$.

Suppose $\rho' \in Y_p \cup Y_c$. Then $H' = H$, so $\neg(H \# H')$. Moreover, since $c, c' \in \underline{e} \cup {}^\bullet e$, and H is a history for e, we have $c, c' \in \mathsf{Cut}(H)$. Therefore, $\rho \parallel \rho'$ as desired.

Let us now assume that the right-hand part of the disjunction holds, and assume by contradiction that $\neg(\rho \parallel \rho')$. Then either $H \# H'$ or $c, c' \notin \mathsf{Cut}(H \cup H')$.

1. If $H \# H'$, then (i) either there exists $e_1 \in H$, $e_2 \in H' \setminus H$ with $e_2 \nearrow e_1$ or (ii) $e_1 \in H \setminus H'$, $e_2 \in H'$ with $e_1 \nearrow e_2$. In either case, $e_1 = e$ must hold; in fact, if e_1 were in $H \setminus \{e\}$, then $\rho_i \not\parallel \rho'$ for some $\rho_i \in X_p \cup X_c$.
   (i) There are three cases for $e_2 \nearrow e$.
   - If $e_2 < e$, then $e_2 \in H$ because H is causally closed, which contradicts $e_2 \in H' \setminus H$.
   - If $\underline{e_2} \cap {}^\bullet e \neq \emptyset$, then again $e_2 \in H$ because ${}^\bullet \underline{e} \cap H' \subseteq H$.
   - If ${}^\bullet e_2 \cap {}^\bullet e \neq \emptyset$, then clearly $\rho' \not\parallel \rho_i$ for some $\rho_i \in X_p$.

   (ii) There are three cases for $e \nearrow e_2$.
   - If $e < e_2$, then H' consumes all tokens from ${}^\bullet e$, so ρ' is not concurrent with any element of X_p.
   - If $\underline{e} \cap {}^\bullet e_2 \neq \emptyset$, then ρ' is not concurrent with some element of X_c.
   - If ${}^\bullet e \cap {}^\bullet e_2 \neq \emptyset$, see (i).
2. If $c' \notin \mathsf{Cut}(H \cup H')$, then there exists $e_1 \in H$ with $c' \in {}^\bullet e_1$. If $e_1 \neq e$, we would get $\rho_i \not\parallel \rho'$ for some $\rho_i \in X_p \cup X_c$. But if $e_1 = e$, then $c' \in {}^\bullet e$, contradicting our assumption.
3. If $c \notin \mathsf{Cut}(H \cup H')$, then there exists $e_1 \in H'$ with $c \in {}^\bullet e_1$. If $\rho \in Y_p$, then H' consumes all of ${}^\bullet e$; if $\rho \in Y_c$, then H' consumes some element of $\underline{e}$. In either case, we get non-concurrency between ρ' and some element of $X_p \cup X_c$. □

As a complement to Proposition 4, the following result allows to compute the concurrency relation for compound conditions.

Proposition 5 (updating concurrency - compound). *Let $\rho = \langle c, H_1 \cup H_2 \rangle$ be a compound condition of $\mathcal{E}$, where $\rho_1 = \langle c, H_1 \rangle$, $\rho_2 = \langle c, H_2 \rangle$ are enriched conditions verifying $\neg(H_1 \# H_2)$. Let $\rho' = \langle c', H' \rangle \in \mathcal{E}$ be any enriched condition. Then*

$$\rho \parallel \rho' \iff \rho_1 \parallel \rho' \wedge \rho_2 \parallel \rho'$$

Proof Let $H = H_1 \cup H_2$.
Left-to-right.

By contradiction, assume $\rho \parallel \rho'$ and w.l.o.g. $\rho_1 \not\parallel \rho'$. Then one of the four statements in Remark 4 must be false:

1. There exist $e_1 \in H'$ and $e_2 \in H_1 \setminus H'$ verifying $e_2 \nearrow e_1$. As $e_2 \in H \setminus H'$, we have $H \# H'$, a contradiction to $\rho \parallel \rho'$.

2. There exist $e_1 \in H_1$ and $e_2 \in H' \setminus H_1$ verifying $e_2 \nearrow e_1$. As $H_1 \subseteq H$, we have $e_1 \in H$. Regarding e_2, we have two cases: either $e_2 \notin H$ or $e_2 \in H$. Assuming the former immediately leads us to the contradiction $H \# H'$. Assuming $e_2 \in H = H_1 \cup H_2$ leads to $e_2 \in H_2 \setminus H_1$. In turn, this implies $H_1 \# H_2$, a contradiction to our hypothesis.
3. There exists $e \in H_1$ such that $c' \in {}^\bullet e$. Then $e \in H$ and H also consumes c', a contradiction to $\rho \parallel \rho'$.
4. There exists $e \in H'$ such that $c \in {}^\bullet e$. This is a contradiction to $\rho \parallel \rho'$.

Right-to-left. Assume $\rho_1 \parallel \rho'$, $\rho_2 \parallel \rho'$, and by contradiction $\rho \not\parallel \rho'$. We consider the four cases of Remark 4:

1. There exist $e_1 \in H$ and $e_2 \in H' \setminus H$ verifying $e_2 \nearrow e_1$. Then either $e_1 \in H_1$, and $H_1 \# H'$ holds, or $e_1 \in H_2$ and $H_2 \# H'$ holds. In any case we reach a contradiction to our hypothesis.
2. There exist $e_1 \in H'$ and $e_2 \in H \setminus H'$ verifying $e_2 \nearrow e_1$. Same argument as before, regarding e_2 instead of e_1.
3. There exists $e \in H$ such that $c' \in {}^\bullet e$. Either $e \in H_1$ and $\neg(\rho_1 \parallel \rho')$ or $e \in H_2$ and $\neg(\rho_2 \parallel \rho')$. In any case we reach a contradiction to our hypothesis.
4. There exists $e \in H'$ such that $c \in {}^\bullet e$. This is a contradiction to $\rho_1 \parallel \rho'$. □

5.3. Unique possible extensions

Propositions 2 and 3 show how possible extensions are constructed, in both lazy and eager fashions. Essentially, a history H for an event is constructed by taking *generating* histories for the conditions in $\underline{e}$, while for the conditions in ${}^\bullet e$ one takes a generating history and optionally some *reading* histories. In the eager case, the latter are combined to one single compound history.

The optionality of the reading histories means that, in some cases, the same history H may be constructed in different ways, by combining different sets of enriched conditions. Consider the unfolding in Fig. 7. Condition c has $n+1$ different reading histories: $H_0 := \emptyset$, $H_1 := \{e_1\}$, …, $H_n := \{e_1, \ldots, e_n\}$, while c' has one single history $H := H_n$. Notice that we have $\langle c, H_i \rangle \parallel \langle c', H \rangle$ for all $i = 0, \ldots, n$. Thus, there exists a multitude of possibilities to construct the enriched event $\langle e, H \cup \{e\} \rangle$: there are 2^{n+1} collections satisfying Proposition 2 and $n+1$ collections for Proposition 3.

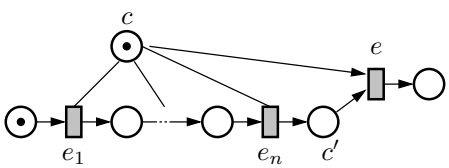

Figure 7: Propositions 2 and 3 allow multiple constructions of $\langle e, \{e_1, \ldots, e_n, e\} \rangle$.

In the following, we discuss how to remove this ambiguity, i.e. how additional constraints can be inserted into Propositions 2 and 3 so that every tuple $\langle e, H\rangle$ can be obtained from a unique collection of enriched conditions. Roughly, the idea is simple: if one element of $X_p \cup X_c$ contains an event e' that reads from $c \in {}^\bullet e$, then that event must be contained in a reading (resp. compound) condition for c included in X_p.

In both lazy and eager mode, this requires to compute an additional relationship between enriched conditions, and we propose how this can be computed.

5.3.1. Lazy approach: Subsumption

For the lazy approach, which deals exclusively with generating and reading histories, we use the notion of subsumption:

Definition 15 (subsumption). *Let $\rho = \langle c, H\rangle$ be a generating or reading condition and $\rho' = \langle c', H'\rangle$ be a reading condition, where H' is the history of some $e \in \underline{c'}$. If $e \in H$, $c' \in \mathsf{Cut}(H)$, and $H' = H[\![e]\!]$, we say that ρ* subsumes *ρ', written $\rho \propto \rho'$.*

In other words, the subsuming condition ρ includes H' and reads but never consumes c'. For instance, in Fig. 7, $\langle c', H\rangle \propto \langle c, H_i\rangle$, for all $i = 1, \ldots, n$.

Definition 16 (subsumption closure). *Let X_p, X_c be sets of enriched conditions enjoying the properties in Proposition 2. We call X_p, X_c* subsumption-closed *if additionally for any $\rho \in X_p \cup X_c$, if $\rho \propto \rho'$ for some reading condition ρ' such that ${\rho'}^\uparrow \in X_p$, then $\rho' \in X_p$.*

For instance, the collection $\langle c, \emptyset\rangle, \langle c', H\rangle$ satisfies the conditions of Proposition 2 but is not subsumption-closed since $\langle c', H\rangle \propto \langle c, H_1\rangle$, $\langle c, H_1\rangle^\uparrow = \langle c, \emptyset\rangle \in X_p$ but $\langle c, H_1\rangle \notin X_p$. The only subsumption-closed collection to produce $\langle e, H \cup \{e\}\rangle$ is $\{\, \langle c, H_i\rangle \mid i = 0, \ldots, n \,\} \cup \{\langle c', H\rangle\}$.

We now show how, for subsumption-closed collections X_p, X_c, knowledge of the relation $\propto$ can be used for computing concurrency.

Proposition 6 (concurrency vs. subsumption). *In Algorithm 1, let $\mathcal{E}$ be the current EP, where $\langle e, H\rangle$ is the last addition thanks to sets X_p, X_c as per Proposition 2 and assume X_p, X_c* subsumption-closed*. We denote by $Y_p = e^\bullet \times \{H\}$ and $Y_c = \underline{e} \times \{H\}$ the generating and reading conditions created by the addition of $\langle e, H\rangle$. Let $\rho' = \langle c', H'\rangle$ be any enriched condition such that $\rho' \notin Y_p \cup Y_c$, $c' \notin {}^\bullet e$, and $\forall \rho_1 \in X_p \cup X_c \colon (\rho_1 \parallel \rho')$. Then the following are equivalent:*

1. *for all ρ'' such that $\rho' \propto \rho''$ and ${\rho''}^\uparrow \in X_p$ we have $\rho'' \in X_p$;*
2. *$\underline{{}^\bullet e} \cap H' \subseteq H$.*

Proof *Left-to-right*

Let e'' be in $\underline{{}^\bullet e} \cap H'$, i.e. $e'' \in H'$ and there is $c_1 \in {}^\bullet e \cap \underline{e''}$. Let $\rho'' = \langle c_1, H'[\![e'']\!]\rangle$ and note that ρ' does not consume c_1, so by definition $\rho' \propto \rho''$. Let

$\rho_1 = \langle c_1, H_1 \rangle \in X_p$ be the generating condition associated with c_1 in X_p. Denote $\rho''^{\uparrow} = \langle c_1, H_1'' \rangle$. Recall that $\rho_1 \parallel \rho'$ implies the existence of a configuration $\mathcal{C}$ such that $H_1 \sqsubseteq \mathcal{C}$ and $H_1'' \sqsubseteq H'[\![e'']\!] \sqsubseteq H' \sqsubseteq \mathcal{C}$. Therefore $\neg(H_1'' \# H_1)$, and thus, by Remark 5, $H_1'' = H_1$. Therefore, $\rho_1 = \rho''^{\uparrow}$ and thus, by 1. we have $\rho'' \in X_p$ and hence $e'' \in H$.

Right-to-left

Suppose that $\rho' \propto \rho'' = \langle c_1, H_1'' \rangle$, where $H_1'' = H'[\![e'']\!]$ for some $e'' \in H'$, and suppose $\rho''^{\uparrow} = \rho_1 = \langle c_1, H_1 \rangle \in X_p$. Now $c_1 \in {}^\bullet e \cap \underline{e''}$, and since, by construction of the unfolding, ${}^\bullet e \cap \underline{e} = \emptyset$ (see Definition 1), we have $e'' \neq e$. But $e'' \in {}^\bullet \underline{e} \cap H'$ and by 2. we get $e'' \in H$, so there must be some $\rho_2 = \langle c_2, H_2 \rangle \in X_p \cup X_c$ with $e'' \in H_2$. By assumption, $\rho_2 \parallel \rho'$, so there exists some configuration $\mathcal{C}$ such that $H_2[\![e'']\!] \sqsubseteq H_2 \sqsubseteq \mathcal{C}$ and $H_1'' = H'[\![e'']\!] \sqsubseteq H' \sqsubseteq \mathcal{C}$. This implies $\neg(H_2[\![e'']\!] \# H_1'')$, hence by Remark 5 they are equal, so by definition $\rho_2 \propto \rho''$. If X_p, X_c is subsumption-closed, then we have $\rho'' \in X_p$, as desired. □

Proposition 6 shows how knowledge of the subsumption relation can be useful for updating the concurrency relation; condition 1 of Proposition 6 can be used to implement or replace the condition ${}^\bullet \underline{e} \cap H' \subseteq H$ of Proposition 4. This, on the other hand, begs for a way to incrementally compute $\propto$, too, which is done by Proposition 7.

Proposition 7 (updating subsumption). *In Algorithm 1, let $\mathcal{E}$ be the current EP, where $\langle e, H \rangle$ is the last addition thanks to sets X_p, X_c as per Proposition 2. We denote by $Y_p = e^\bullet \times \{H\}$ and $Y_c = \underline{e} \times \{H\}$ the generating and reading conditions created by the addition of $\langle e, H \rangle$. Let $\rho = \langle c, H \rangle \in Y_p \cup Y_c$, and let $\rho' = \langle c', H' \rangle \in \mathcal{E}$ be any other enriched condition. Then $\rho \propto \rho'$ if and only if*

$$\rho' \in Y_c \ \vee \ (\exists \rho'' \in X_p \cup X_c \colon \rho'' \propto \rho' \wedge \rho \parallel \rho')$$

Proof *Left-to-right.*

If $\rho \propto \rho'$, then by Definition 15 there is some $e' \in H \cap \underline{c'}$ such that $H' = H[\![e']\!]$ and $c' \in \mathsf{Cut}(H)$.

1. Either $e' = e$, then $H' = H$ and $\rho' \in Y_c$.
2. Otherwise e' is contained in some H'' such that $\rho'' = \langle c'', H'' \rangle \in X_p \cup X_c$. Since $c' \in \underline{e'}$, c' must be either initial or produced by H'', and H'' does not consume c' (since H does not), so $c' \in \mathsf{Cut}(H'')$.
   We have $H' = H[\![e']\!] \sqsubseteq H$ and $H''[\![e']\!] \sqsubseteq H'' \sqsubseteq H$. Therefore, $\neg(H' \# H''[\![e']\!])$ and by Remark 5 we have $H' = H''[\![e']\!]$ and hence $\rho'' \propto \rho'$.
   Finally, $\rho \parallel \rho'$ follows from $\rho \propto \rho'$: we have $H' \sqsubseteq H$, so clearly $\neg(H \# H')$, and $c, c' \in \mathsf{Cut}(H)$.

Right-to-left.

1. If $\rho' \in Y_c$, then $c' \in \underline{e}$ and $H' = H = H[\![e]\!]$, so clearly $\rho \propto \rho'$ (and vice versa).

2. Let $\rho'' = \langle c'', H'' \rangle \in X_p \cup X_c$ such that $\rho'' \propto \rho'$ and assume $\rho \parallel \rho'$. From the former we get $c' \in \mathsf{Cut}(H'')$ and from the latter $c' \in \mathsf{Cut}(H \cup H')$. So c' is initial or produced by $H'' \sqsubseteq H$ and not consumed by H, thus $c' \in \mathsf{Cut}(H)$. Now, $\rho'' \propto \rho'$ implies some $e' \in H''$ such that $H' = H''[\![e']\!]$. It remains to show that $H[\![e']\!] = H''[\![e']\!]$, which again follows from Remark 5. □

The results of this section lead us to the following characterisation of unique possible extensions.

Corollary 1 (unique lazy extensions). *The pair $\langle e, H \rangle$ with $f(e) = t$ is an enriched event iff there exist sets X_p, X_c of enriched conditions such that*

1. *$f(X_p) = {}^\bullet t$ and $f(X_c) = \underline{t}$;*
2. *X_p contains generating or reading conditions, X_c generating conditions;*
3. *$X_p \cup X_c$ contains exactly one generating condition for every $c \in ({}^\bullet e \cup \underline{e})$;*
4. *for all $\rho, \rho' \in X_p \cup X_c$ we have $\rho \parallel \rho'$;*
5. *X_p, X_c are subsumption closed;*
6. *finally, $H = \{e\} \cup \bigcup_{\langle c, H' \rangle \in X_p \cup X_c} H'$.*

Moreover, for any enriched event $\langle e, H \rangle$ there exists exactly one pair of sets X_p, X_c satisfying properties 1–6.

Proof The "iff" part follows almost directly from Proposition 2. We just need to observe that any collection that does not satisfy property 5 can be made subsumption-closed by adding all the ρ' according to the rule given in Definition 16.

It remains to show the uniqueness of the pair X_p, X_c w.r.t. $\langle e, H \rangle$. We start by observing that for any $\rho_1 = \langle c_1, H_1 \rangle \in X_p \cup X_c$, it holds $H_1 \sqsubseteq H$. In fact, let $e' \in H_1$ and $e'' \in H$, such that $e'' \nearrow e'$. Then $e'' \neq e$, otherwise $e' \nearrow^* e \nearrow e'$ would be a cycle of asymmetric conflict in H. Hence there is some $\rho_2 = \langle c_2, H_2 \rangle \in X_p \cup X_c$ such that $e'' \in H_2$. Since $\rho_1 \parallel \rho_2$ and thus $\neg(H_1 \# H_2)$, we deduce $e'' \in H_1$, as desired.

Now, let X'_p, X'_c be another pair of sets of enriched conditions satisfying properties 1–6 above for $\langle e, H \rangle$. For any condition $\rho_1 = \langle c_1, H_1 \rangle \in X_p \cup X_c$, we distinguish two cases. First, if ρ_1 is a generating condition, then by property 3 there exists $\rho'_1 = \langle c_1, H'_1 \rangle \in X'_p \cup X'_c$. By the observation above, $H_1, H'_1 \sqsubseteq H$, hence $\rho_1 \parallel \rho'_1$ and thus, by Remark 5, $\rho_1 = \rho'_1 \in X'_p \cup X'_c$. The second possibility is that ρ_1 is a reading condition, i.e., H_1 is the history of some $e_1 \in {}^\bullet \underline{e}$. Since $e_1 \in H$, there exists $\rho'_2 = \langle c_2, H'_2 \rangle \in X'_p \cup X'_c$ such that $e_1 \in H'_2$. Clearly $c_1 \in \mathsf{Cut}(H'_2)$ and thus $\rho'_2 \propto \langle c_1, H'_2[\![e_1]\!] \rangle =: \rho'_1$. Moreover, it is easy to see that ${\rho'_1}^\uparrow \in X'_p$ and thus subsumption closure of X'_p, X'_c leads us to conclude that $\rho'_1 \in X'_p$. To conclude, observe that $H'_2[\![e_1]\!] \sqsubseteq H'_2 \sqsubseteq H$ and $H'_1 \sqsubseteq H$, and thus, by Remark 5, $H'_2[\![e_1]\!] = H_1$, since they are both histories of event e_1. Therefore $\rho_1 = \rho'_1 \in X'_p \cup X'_c$.

Therefore $X_p \cup X_c \subseteq X'_p \cup X'_c$. By symmetry we conclude that they must coincide. □

5.3.2. Eager approach: Asymmetric concurrency

We now show how possible extensions can be made unique in the eager approach. In the lazy case, we used the concept of subsumption to achieve this. Recall its intuition: if one element of $X_p \cup X_c$ contains an event e' that reads from some $c \in {}^\bullet e$, then that event must be contained in a reading condition for c included in X_p. For the eager case, this idea must be adapted to compound conditions, where the history of c may be a union of several of its readers. In this case, we demand that at least those readers of c contained elsewhere in $X_p \cup X_c$ are included in the (compound) history chosen for c in X_p.

We introduce a new relation between enriched conditions that captures this intuition. It is a refinement of $\parallel$ that we call *asymmetric concurrency* ($/\!/$). It turns out that unique possible extensions can be characterised using only this relation.

Definition 17 (asymmetric concurrency). *Let $\rho = \langle c, H\rangle$ and $\rho' = \langle c', H'\rangle$ be two enriched conditions. We say that ρ is* asymmetrically concurrent *to ρ', written $\rho \,/\!/\, \rho'$ iff $\rho \parallel \rho'$ and $\underline{c} \cap H' \subseteq H$.*

Notice that $/\!/$ is an asymmetric relation.

The following proposition relates $\parallel$ and $/\!/$ for generating conditions:

Proposition 8 (concurrency vs asymmetric concurrency). *Suppose that $\rho = \langle c, H\rangle$ is a generating enriched condition and $\rho' = \langle c', H'\rangle$ is an arbitrary enriched condition. Then $\rho \parallel \rho'$ iff $\rho \,/\!/\, \rho'$ or $\rho' \,/\!/\, \rho$.*

Proof We only need to prove the direction from left to right, the other trivially follows from Definition 17. So suppose by contradiction that $\rho \parallel \rho'$ and neither $\rho \,/\!/\, \rho'$ nor $\rho' \,/\!/\, \rho$. Thus, there exist $e_1 \in (\underline{c} \cap H') \setminus H$ and $e_2 \in (\underline{c'} \cap H) \setminus H'$. So H is not empty, and it is in fact the history of some event $e \in {}^\bullet c$. So $e < e_1$, therefore e must be in H'. Moreover, $e_2 \in H \setminus H'$, so $e_2 \neq e$ and $e_2 \nearrow^+ e$. This means $H \# H'$, contradicting $\rho \parallel \rho'$. □

As for the computation of $/\!/$, notice that $\langle c, H\rangle \,/\!/\, \langle c', H'\rangle$ implies $\langle c, H\rangle \parallel \langle c', H'\rangle$. We can thus reuse Propositions 4 and 5 to obtain $\parallel$ and check $\underline{c} \cap H' \subseteq H$ and $\underline{c'} \cap H \subseteq H'$ at the same time. More details are discussed in Section 6.

The following Corollary 2 summarises the results of this section, showing how unique possible extensions can be characterised using only $/\!/$.

Corollary 2 (unique eager extensions). *The pair $\langle e, H\rangle$ with $f(e) = t$ is an enriched event iff there exist sets X_p, X_c of enriched conditions such that*

1. *$f(X_p) = {}^\bullet t$ and $f(X_c) = \underline{t}$;*
2. *X_p contains arbitrary enriched conditions, X_c generating conditions;*
3. *$X_p \cup X_c$ contains exactly one enriched condition for every $c \in ({}^\bullet e \cup \underline{e})$;*
4. *$\rho \,/\!/\, \rho'$ for $\rho \in X_p$ and $\rho' \in X_p \cup X_c$;*
5. *$\rho \,/\!/\, \rho'$ or $\rho' \,/\!/\, \rho$ for all $\rho, \rho' \in X_c$;*
6. *finally, $H = \{e\} \cup \bigcup_{\langle c, H'\rangle \in X_p \cup X_c} H'$.*

Moreover, for any enriched event $\langle e, H\rangle$ there exists exactly one pair of sets X_p, X_c satisfying properties 1–6.

Proof The "iff" follows almost directly from Proposition 3 and Proposition 8. In particular, properties 4 and 5 imply that $\rho \parallel \rho'$ holds for all $\rho, \rho' \in X_p \cup X_c$. Moreover, let X_p, X_c be some collection satisfying the conditions of Proposition 3 but not Corollary 2. Then there are $\rho_1 = \langle c_1, H_1\rangle \in X_p$ and $\rho' = \langle c_2, H_2\rangle \in X_p \cup X_c$ with $\rho_1 \parallel \rho_2$ and some $e' \in (\underline{c_1} \cap H_2) \setminus H_1$. We have $H_2[\![e']\!] \sqsubseteq H$ and therefore $\neg(H_1 \# H_2[\![e']\!])$. Thus ρ_1 can be replaced with the compound condition $\rho_1' = \langle c_1, H_1 \cup H_2[\![e']\!]\rangle$ in X_p. This process can be repeated until property 4 is satisfied.

It remains to show uniqueness. Suppose there exists another collection X_p', X_c' for the same enriched event $\langle e, H\rangle$. There must be some $\rho_1 = \langle c, H_1\rangle \in X_p \cup X_c$ and $\rho_2 = \langle c, H_2\rangle \in X_p' \cup X_c'$ with $H_1 \neq H_2$ and w.l.o.g. some $e_1 \in H_1 \setminus H_2$. Since $e_1 \in H$, there must be some $\rho_3 = \langle c', H_3\rangle \in X_p' \cup X_c'$ such that $e_1 \in H_3$.

If $c \in {}^\bullet e$ and $e_1 \in \underline{c}$, then $\rho_2 \in X_p$. but $\rho_2 \mathbin{//} \rho_3$ would be violated.

So let $c \in \underline{e} \cup {}^\bullet e$. If $e_1 \in {}^\bullet c$, then H_2 would not be a history of c. The final possibility is that $e_1 \nearrow^+ e'$ for some $e' \in {}^\bullet c \cup \underline{c})$ and $e' \in H_2$. But then $H_2 \# H_3$. □

5.4. Discussion: lazy vs eager approach

In order to discover possible extensions of the form $\langle e, H\rangle$, both approaches consider combinations of generating and reading histories for conditions $c \in {}^\bullet e$.

Consider Proposition 2. For every possible extension, the lazy approach takes one generating and possibly several reading histories for c, all of which must be concurrent. If the events in $\underline{c}$ have many different histories, or $\underline{c}$ is large, then many different combinations need to be checked for concurrency.

The eager approach (Proposition 3) takes exactly one enriched condition of arbitrary type, including compound, for c. A compound history is a set of concurrent reading histories (Definition 12); thus a compound condition represents pre-computed information needed to identify possible extensions. In this sense, the eager and the lazy approach can be thought of as different time/space tradeoffs.

We consider two examples in which eager beats lazy and vice versa. In Fig. 8 (a), condition c has a sequence of n readers and hence $n+1$ histories $\{e_1, \ldots, e_i\}$, for $i = 0, \ldots, n$. For each history H of c', eager simply combines H with the $n+1$ histories for c, while lazy checks all 2^n subsets of $e_1, \ldots, e_n$ to find these $n+1$ compound histories. If c' has many histories, eager becomes largely superior. Of course, an intelligent strategy may help lazy to avoid exploring all 2^n subsets one by one. However, even with a good strategy, lazy still has to enumerate at least the same combinations as eager; and since the problem of identifying the useful subsets is NP-complete [20], there will always be instances where lazy becomes inefficient, whatever strategy is employed.

On the other hand, consider Fig. 8 (b). Again, c has n readers, this time yielding 2^n histories. Suppose that $f(c)$ is an input place of some transition t.

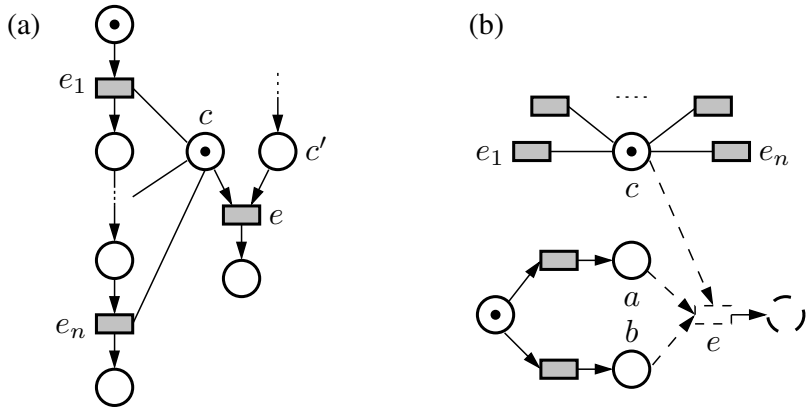

Figure 8: Good examples for the eager (a) and the lazy (b) approach.

Now, if t also has $f(a)$ and $f(b)$ in its preset, then no t-labelled event e will ever be generated in the unfolding, and all histories of c are effectively useless. Since those compound conditions also appear in the computation of the concurrency relation, they become a liability in terms of both memory and execution time. The lazy approach does not suffer from this problem here.

Both approaches therefore have their merits, and we implemented them both. We shall report on experiments in Section 7.

5.5. Memory usage

We shall briefly discuss the memory usage arising from the methods proposed in this section. There are two major factors determining memory consumption:

- As some of our examples show, notably Fig. 3 (a) and Fig. 8 (b), a condition in the unfolding may have a number of histories exponential in the number of events reading from it. Thus, the memory usage for creating a finite unfolding prefix may be exponentially larger than the unfolding prefix that is eventually produced.

- Moreover, the memory needed to store the binary relations between enriched conditions such as $\|$, $\propto$, and $//$ is quadratic w.r.t. the number of these enriched conditions in the worst case.

One could ask whether these memory blowups are really necessary. Let us first discuss this question with regard to the second point: in a concrete implementation, the binary relations $\|$, $\propto$, and $//$ could either be stored explicitly (at the cost of quadratic memory overhead) or decided individually for each pair whenever necessary, by directly checking the respective definitions, at the cost of higher computation time. We initially implemented the second approach [21]; however, the running times were such that only small unfoldings with a few hundred events could be produced in reasonable time. We therefore chose to store the binary relations explicitly.

Histories, on the other hand, allow to easily identify new possible extensions (as per Propositions 2 and 3). In principle, one could imagine an additional time/space tradeoff in which not only the concurrency relation but even the histories themselves are constructed on demand whenever one tries to instantiate the above-mentioned propositions, leading to an algorithm which consumes only linear space w.r.t. the size of the unfolding prefix. Due to the unsatisfactory results with the concurrency relation, we did not consider this approach.

In any case, the memory usage is asymptotically the same as for the PR-encoding. Section 6 contains some hints on how to store histories efficiently, and Section 7 provides data on actual memory usage on several examples.

6. Efficient prefix construction

We implemented the procedure from Algorithm 1, using the methods proposed in Section 5. The resulting tool, called Cunf, is publicly available [11]. Cunf expects as input a c-net and produces as output a complete unfolding prefix. The current implementation of Cunf is restricted to safe c-nets because our examples of interest are in this domain. Moreover, this choice simplifies certain data structures and algorithms in the implementation.

Notice that there exist efficient tools for the unfolding of Petri nets, such as Mole [12] or Punf [19]. While we profited much from the experiences gained from developing Mole, Cunf is not an extension of it. The issues of asymmetric conflict and histories permeate every aspect of the construction so that we went for a completely new implementation in C, comprising some 4,000 lines of code.

Here, we review some features such as data structures and implementation details relevant to handling the complications imposed by contextual unfoldings, that helped to produce an efficient tool. Experiments are reported in Section 7.

6.1. The history graph

Cunf needs to maintain enriched events and conditions, i.e. tuples $\langle e, H\rangle$ or $\langle c, H\rangle$, where H is a history. We store them in a graph structure, that grows whenever the enriched prefix $\mathcal{E}$ is extended. Formally, the *history graph* associated with $\mathcal{E}$ is a directed graph $\mathcal{H}_\mathcal{E}$ whose nodes are the enriched events of $\mathcal{E}$, and with edges $\langle e, H\rangle \to \langle e', H'\rangle$ iff $e' \in H$ and $H' = H[\![e']\!]$ and either (i) $(e'^\bullet \cup \underline{e'}) \cap {}^\bullet e \neq \emptyset$ or (ii) $e'^\bullet \cap \underline{e} \neq \emptyset$. Each node $\langle e, H\rangle$ is labelled by e.

Intuitively, $\mathcal{H}_\mathcal{E}$ has an edge between two enriched events $\langle e, H\rangle$ and $\langle e', H'\rangle$ iff some enriched condition $\langle c, H'\rangle$ was used to construct $\langle e, H\rangle$ (in the sense of Proposition 2 or Proposition 3).

This structure allows Cunf to perform many operations efficiently: every additional enriched event enlarges the graph by just one node plus some edges; common parts of histories are shared. We can easily enumerate the events in $H \in \chi(e)$ by following the edges from node $\langle e, H\rangle$, and $\mathcal{H}_\mathcal{E}$ implicitly represents the relation $\sqsubset$. Given an event e, we can enumerate the histories in $\chi(e)$ by keeping the list of nodes in $\mathcal{H}_\mathcal{E}$ that are labelled by e. Given a condition c, we can enumerate its generating and reading histories similarly.

Compound conditions are stored in a shared-tree-like structure, where leaves represent reading histories and internal nodes compound histories. An internal node has two children, one of which is a leaf, the other either internal or a leaf. One easily sees that a compound history of c corresponds, w.l.o.g., to a union $H_1 \cup \cdots \cup H_n$ of reading histories. Every internal node represents such a union, and the structure allows sharing if one compound history contains another.

6.2. Possible extensions

Cunf behaves similar to Mole or other unfolders in its flow of logic, but its actions are on enriched events and conditions. We start with a prefix containing just $\widehat{m}_0$ and identify the initial possible extensions. As long as the set of possible extensions is non-empty, we choose a "minimal" extension and add it unless it is a cutoff. For "minimal", we use the adequate order $\prec_F$ from [10]. Adding $\langle e, H\rangle$ means adding H to $\chi(e)$, creating e first if necessary. The addition of $\langle e, H\rangle$ will give rise to various types of enriched conditions for whom we compute the concurrency relation (see below). Whenever we add an enriched condition ρ, we attempt to find possible extensions, i.e. sets X_p, X_c matching the conditions in Propositions 2 and 3 such that $X_p \cup X_c$ includes ρ, where, in order to implement condition 4, we use the precomputed binary concurrency relation. Upon identifying a possible extension $\langle e, H\rangle$, we immediately compute its marking, information relevant to deciding $\prec_F$, and certain lists $r(H), s(H)$ during two linear traversals of H. Details on $r(H)$ and $s(H)$ are given below.

6.3. Concurrency relation

The relation $\|$ on the enriched conditions of $\mathcal{E}$ can be stored and updated whenever new possible extensions are appended to $\mathcal{E}$. We detail now how Propositions 4 and 5 are used to efficiently compute this update.

Let $\mathsf{c}(\rho)$ denote the set of enriched conditions ρ' verifying $\rho \parallel \rho'$. The relation $\|$ is generally sparse, and Cunf stores $\mathsf{c}(\rho)$ as a list. However, for the purpose of the following, $\mathsf{c}(\rho)$ could also be a row in a matrix representing $\|$.

For reading and generating conditions ρ (Proposition 4), Cunf initially sets $\mathsf{c}(\rho)$ to $Y_p \cup Y_c$. Next, it computes the intersection of $\mathsf{c}(\rho')$ for all $\rho' \in X_p \cup X_c$, and filters out those $\langle c', H'\rangle$ for which $\underline{{}^\bullet e} \cap H' \not\subseteq H$ holds. In order to compute this condition without actually traversing H and H', we use the sets $r(H)$ and $s(H)$ computed earlier (see above). These are defined as $r(H) := \{\, e' \in H \mid \underline{e'} \cap \mathsf{Cut}(H) \neq \emptyset \,\}$ and $s(H) := \{\, e' \in H \mid e' \in \underline{{}^\bullet e} \,\}$. Then $\underline{{}^\bullet e} \cap H' \not\subseteq H$ holds iff $\underline{{}^\bullet e} \setminus s(H) \cap r(H') \neq \emptyset$, which can be computed traversing $\underline{{}^\bullet e}$ and $s(H)$ one time, and checking $r(H')$ for every ρ'. Note that, while the other steps have their counterparts in Petri net unfoldings, this step is new and specific to c-nets. However, we find that this implementation keeps the overhead very small. The checks to test $\underline{c} \cap H' \subseteq H$ required to compute // are done similarly and can in fact be combined with the aforementioned test. We store // inside the lists $\mathsf{c}(\rho)$ that represent $\|$: the lowest two bits of a pointer are "abused" to store whether $\rho \mathbin{//} \rho'$ holds and vice versa.

As for compound conditions ρ built using ρ_1 and ρ_2 (Proposition 5), Cunf computes $\mathsf{c}(\rho)$ as the intersection of $\mathsf{c}(\rho_1)$ and $\mathsf{c}(\rho_2)$.

Certain enriched conditions $\rho = \langle c, H \rangle$ need not to be included in the concurrency relation. It is safe, for instance, to leave $\mathsf{c}(\rho)$ empty if ρ is generating and $f(c)^\bullet \cup \underline{f(c)} = \emptyset$, or if H is a cutoff. We can also avoid computing $\mathsf{c}(\rho)$ if ρ is reading or compound and $f(c)^\bullet = \emptyset$, even if $\underline{f(c)} \neq \emptyset$.

6.4. Splitting the concurrency relation

Let $\rho = \langle c, H \rangle$ be an enriched condition. As mentioned in the previous paragraph, Cunf manages the set $\mathsf{c}(\rho)$ containing the enriched conditions ρ' such that $\rho \parallel \rho'$. We found that the performance of the tool benefits greatly in some cases by splitting $\mathsf{c}(\rho)$ into two sets: $\mathsf{c}_1(\rho) = \{ \langle c, H' \rangle \mid \rho \parallel \langle c, H' \rangle \}$ and $\mathsf{c}_2(\rho) = \mathsf{c}(\rho) \setminus \mathsf{c}_1(\rho)$. In other words, $\mathsf{c}_1(\rho)$ contains the concurrent pairs for the same condition c and $\mathsf{c}_2(\rho)$ the others.

This simple split helps in several places. Suppose, for instance, that ρ is a new enriched condition that we have just added to the prefix.

- If ρ is reading or generating (where H is a history for an event e), we apply Proposition 4 to compute $\mathsf{c}(\rho)$ (cf. Section 6.3). For $\rho' \in X_p$, any $\langle c', H' \rangle \in \mathsf{c}_1(\rho')$ verifies $c' \in {}^\bullet e$, so $\mathsf{c}_1(\rho')$ can be excluded from consideration.

- Next, in the eager approach, we may use ρ to generate compound conditions. For this, we now simply take all $\rho' = \langle c, H' \rangle$ from $\mathsf{c}_1(\rho)$ and create a new compound condition $\rho'' = \langle c, H \cup H' \rangle$. Moreover, $\mathsf{c}_1(\rho'') = \mathsf{c}_1(\rho) \cap \mathsf{c}_1(\rho')$ and $\mathsf{c}_2(\rho'') = \mathsf{c}_2(\rho) \cap \mathsf{c}_2(\rho')$.

- Finally, we may use any new ρ to search for possible extensions according to Propositions 2 and 3 (cf. Section 6.2). In order to find the sets X_p, X_c, we may in certain cases restrict our search to $\mathsf{c}_2(\rho)$ rather than $\mathsf{c}(\rho)$.

7. Experiments

In order to experimentally evaluate our tool, we performed a series of experiments. We were interested in the following questions:

- Is the contextual unfolding procedure efficient?

- What is the size of the unfoldings, compared to Petri net unfoldings?

- How do the various approaches (lazy, eager, PR, plain encoding) compare?

Concerning the second and third point, it is worth noting that we could contrive examples to show arbitrarily large differences between various approaches. As far as the size of the final unfolding is concerned, Fig. 3 already shows that contextual unfoldings may be up to exponentially more succinct than Petri net unfoldings. As far as running time is concerned, Section 5.4 contains examples that would distinguish the eager and the lazy approach in both senses.

To see how the running time of the contextual approaches can be superior to the plain encoding, consider the net in Fig. 9, where transition t reads from two places p_1 and p_2. Both places have an additional reading transition, so

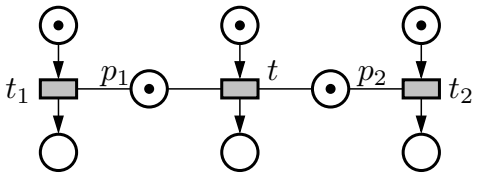

Figure 9: Pairs of independent readers

they each have one (empty) generating history, two reading histories, and one compound history. The contextual unfolding is isomorphic to the net itself. If one expands the context of transition t to k places like p_1 and p_2, then the contextual approaches produce the prefix in time linear to k. The plain encoding, on the other hand, will create an exponential number of events for t, each corresponding to some set of transitions that have previously read from $\underline{t}$.

In order to abstract from such artefacts and get numbers from more realistic examples, we took a set of safe nets that have previously served as benchmarks in the literature on Petri net unfoldings, e.g. [22, 18, 23]. These nets are not specifically geared towards using contextual approaches, though read arcs occur naturally here as part of larger nets. These nets have various characteristics that allowed to test many aspects of our implementation.

For each net N in the set, we first obtained the c-net N' by substituting pairs of arcs (p, t) and (t, p) in N by read arcs. Evidently, the plain encoding of N' is N. Secondly, we obtained the PR-encoding N'' of N'.

We first compared Mole [12] and Cunf on the nets N and N'', which are ordinary Petri nets without read arcs. The object of this exercise was to establish whether Cunf was working reasonably efficient on known examples. Indeed, its running times were always within 70% and 140% of those of Mole, the differences due to minor implementation choices. To abstract from these details, we used Cunf for all further comparisons.

We then used Cunf to produce complete unfoldings of the plain net N, the PR-encoding N'', and of N' using both lazy and eager methods and the order $\prec_F$ from [10]. Table 1 summarises the results.[2]

The columns in the table are subdivided into three parts, corresponding to the contextual net, its PR-encoding, and its plain encoding. For contextual nets, we first give the number of events and conditions contained in the complete finite prefix (columns $|E|$ and $|B|$, in thousands). The number for $|E|$ is actually somewhat larger than what is strictly necessary according to Definition 4 because it also includes events that are enabled by cutoff-free configurations even if those events are not part of any non-cutoff enriched event (see also the discussion at the end of Section 3).

[2]Experiments performed using revision 55 of the Cunf tool, compiled with `gcc` 4.4.5. Our machine has twelve 64bit Intel Xeon CPUs, running at 2.67GHz, 50GB RAM and executes Linux 2.6.32-5.

	Contextual unfolding				(Eager)			(Lazy)		PR unfolding						Plain unfolding				
Net	$\|E\|$	$\|B\|$	$\|E_{\text{cut}}\|$	$\|{}^\bullet e\|$	comp.	t_E	m_E	t_L	m_L	$\|E\|$	$\|B\|$	$\|E_{\text{cut}}\|$	$\|{}^\bullet e\|$	t_R	m_R	$\|E\|$	$\|B\|$	$\|E_{\text{cut}}\|$	t_P	m_P
`bds_1.sync`	1.8k	2.8k	40%	1.5	0	0.11	17	1.5	0.9	2.3	4.8	40%	2.3	2.1	1.9	7.0	13.5	67%	4.2	2.5
`byzagr4_1b`	8.0k	17.6k	4%	2.9	72	2.60	163	1.4	1.0	1.0	1.4	4%	1.9	3.4	1.5	1.8	2.4	5%	1.4	1.5
`dpd_7.sync`	10.4k	21.4k	25%	2.1	0	0.98	59	1.2	1.0	1.0	1.4	25%	1.4	1.2	1.4	1.0	1.4	25%	1.0	1.2
`elevator_4`	16.9k	28.6k	44%	1.7	0	1.24	71	1.1	0.9	1.0	13.7	44%	22.4	358	66.3	1.0	1.7	44%	1.8	1.1
`ftp_1.sync`	50.9k	96.6k	26%	1.9	0	24.81	296	0.6	1.0	1.0	1.6	26%	1.6	2.5	6.3	1.8	2.8	37%	2.0	1.9
`furnace_4`	94.4k	147.4k	68%	1.6	0	19.04	266	0.7	0.9	1.1	1.9	70%	1.8	1.7	2.1	1.2	1.8	69%	0.9	1.4
`key_4`	4.8k	7.9k	16%	1.7	23.3k	1.58	110	711	0.4	4.6	5.5	19%	13.4	6.4	0.7	14.6	17.7	46%	0.8	1.1
`mmgt_4.fsa`	46.9k	92.1k	45%	2.0	0	0.97	70	1.1	1.0	1.0	1.0	45%	1.0	1.0	1.0	1.0	1.0	45%	0.9	1.0
`q_1.sync`	10.7k	20.6k	13%	1.9	0	1.36	77	1.0	0.9	1.0	1.5	13%	1.5	2.2	2.1	1.0	1.5	13%	1.0	1.1
`rw_12.sync`	98.4k	196.8k	92%	2.0	0	3.35	171	1.2	1.0	1.0	1.5	92%	1.5	3.4	2.5	1.0	1.5	92%	1.5	1.0
`rw_1w3r`	14.5k	24.2k	32%	1.7	191	0.27	36	1.6	1.0	1.0	1.7	34%	1.7	2.0	1.6	1.1	1.2	34%	0.8	0.9
`dme11`	9.2k	16.7k	1%	1.8	0	7.89	516	0.9	1.0	1.0	1.9	1%	1.9	1.0	0.9	1.0	1.9	1%	0.8	0.9
`rw_2w1r`	9.4k	15.3k	15%	1.6	0	0.23	31	1.0	1.0	1.0	4.2	15%	4.7	61.9	6.4	1.0	1.2	15%	0.7	0.9

Table 1: Experimental results. Data for the eager approach are absolute numbers, whereas for lazy, place-replication, and plain unfolding the ratio w.r.t. eager is given; see the text for more information.

The column marked $|E_{cut}|$ provides the percentage of such events; this percentage could be subtracted from $|E|$ to obtain the strictly necessary prefix w.r.t. Definition 4. Our tool does in fact detect those events anyway, so their inclusion hardly affects the running time.

The column $|{}^\bullet e|$ gives the average preset size of an event. The four columns mentioned so far are identical for the eager and the lazy approach. For the eager approach, we then list the number of compound conditions (causing additional memory overhead of the eager approach w.r.t. lazy) and the running time[3] in seconds (t_E) as well as (maximum virtual) memory consumption in megabytes (m_E). For lazy, we list running times (t_L) and memory consumption (m_L) *relative to the eager approach*, i.e. a factor less than 1 means a faster/less memory-consuming computation, and a factor larger than 1 a slower/more memory-consuming one.

For the PR-encoding and the plain encoding, the data for number of events and conditions, running times, memory consumption, and average preset size (only PR) is also given *relative to the eager approach.* We additionally provide the percentage of cutoff events ($|E_{cut}|$). Notice that the number of *enriched* events in lazy and eager equals the number of events in PR (compare the discussion in the introduction). The ratio between number of events in contextual and number of events in PR is thus the average number of histories per event in the contextual approach. We make the following observations:

- We first look at the comparison between lazy and eager. It turns out that in this set of benchmarks, many examples did not exhibit any compound conditions (despite the presence of many read arcs), e.g., because reading actions took place sequentially, or multiple potential readers happened to be in conflict with one another. In those examples, the differences between the two versions are due to the different implementations of the possible extensions (Section 5.1) and the various relations that must be maintained (Section 5.3), sometimes slightly favouring one approach, sometimes the other.

  Significant differences arise where (like in `key_4`) there are many compound conditions; here lazy has some memory savings but performs very badly. An effect to the contrary like in Fig. 8 (b), while in principle possible, did not manifest itself in our benchmarks.

- Compared with PR, the eager approach is consistently more efficient. In several cases (such as `elevator_4` or `rw_2w1r`), PR is orders of magnitudes slower. This clear tendency is slightly surprising given that the enriched contextual prefix has essentially the same size as the prefix of the PR-encoding. We experimentally traced the difference to the enlarged presets of certain transitions in the PR-encoding (see Fig. 3), causing combinatorial overhead and increasing the number of conditions in the concurrency

[3] Actually, the *CPU time.*

relation. Indeed, high running times for PR seem to coincide with high numbers in the $|{}^\bullet e|$ column for PR (recall that this number is relative to the one for contextual).

- Both the eager approach and the plain unfolding handle all examples gracefully. The factors of the running times are between 0.7 and 4.2, meaning eager is between 40% slower and 4 times faster w.r.t. plain. The prefixes produced by the contextual unfolding methods are smaller than in the plain approach in half the cases. Interestingly, these are not always the same as those on which they run faster: for `elevator_4` and `rw_12.sync`, the same number of events is produced more quickly. Here, the read arcs are arranged in such a way that each event still has only one history; the time saving comes from the fact that the contextual approach produces fewer *conditions* and hence a smaller concurrency relation. For `key_4` and `rw_1w3r`, the contextual methods produce smaller unfoldings but take longer to run, due some overhead in the computation of the // relation.

To summarize, this set of benchmarks contained examples where lazy and PR performed badly, whereas eager and plain handled all cases gracefully. The eager approach was the fastest overall, and for all examples its running time was within factor 2 of the fastest approach for that example. The prefixes produced by the contextual methods can be significantly smaller than for their Petri net encodings, which make them suitable candidates for subsequent analysis methods (see Section 8 for a brief discussion).

Moreover, we note that read arcs occur naturally when encoding networks of logic gates as Petri nets, one of the motivations mentioned by McMillan in his seminal paper on the unfolding technique [17]. In this encoding, the signals, i.e. the inputs and outputs of each gate, are modelled with two places for indicating whether the signal is high (1) or low (0). The outputs change as a function of *reading* the inputs. Fig. 10 (a) shows an example of an AND-gate and its encoding as a c-net fragment.

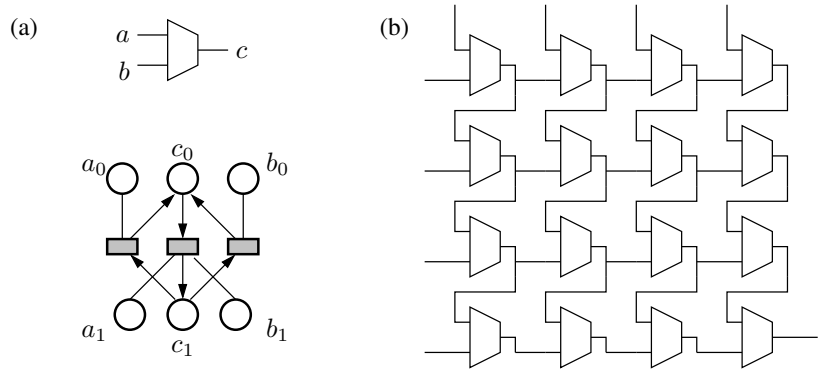

Figure 10: (a) Encoding of a logical AND-gate; and (b) grid of AND-gates

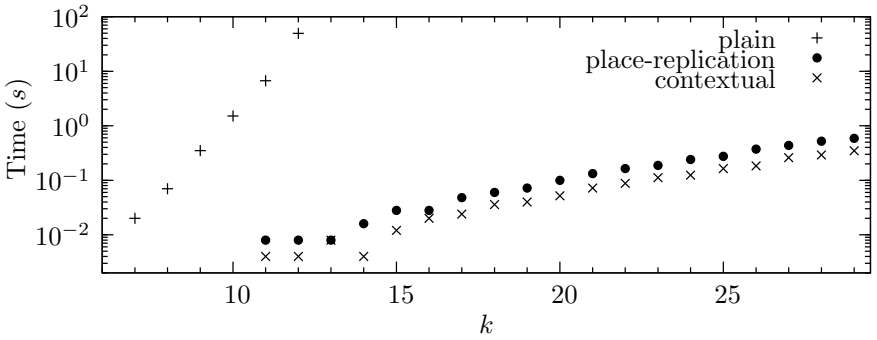

Figure 11: Unfolding times for the plain, PR, and contextual net encodings of the AND-gate networks of size $n := k \times k$.

Various experiments that we conducted indicate that contextual unfoldings perform well on such c-nets. To illustrate the benefits, we present one simple experiment on a particular family of examples, a grid of $n := k \times k$ AND-gates, shown in Fig. 10 (b) for $k = 4$. The inputs for the AND-gates are at the left and top of the figure, and outputs propagate to the right and towards the bottom. Our experiment simulates what happens when the inputs are switched from low to high. Observe that the signal changes required to make the output (at the bottom right) high can occur in many different orders, which are not distinguished by c-net unfoldings. We observed that the plain unfolding of these nets is approximately of size $\mathcal{O}(2.2^{\sqrt{n}})$, while the size of both the contextual and PR prefixes is $\mathcal{O}(n)$. Furthermore, Cunf builds the latter two in $\mathcal{O}(n^3)$, while it requires approximately $\mathcal{O}(5^{\sqrt{n}})$ time to produce the plain unfolding, see Fig. 11.

8. SAT-based property checking of c-nets

While the focus of this article is on the construction of prefixes rather than on their use in verification, we shall briefly sketch how complete prefixes allow to encode properties of c-nets in propositional logic. For this, we adapt the SAT-based reductions of [1], which were presented for Petri nets, to c-nets.

We start by recalling that the reachability problem for bounded nets is PSPACE-complete, whereas the following problem is NP-complete, see e.g. [1]:

> Given a complete prefix of $\mathcal{U}_N$, where N is a bounded *Petri* net, and a marking m of N, is m reachable in N?

It is straightforward to see that the result extends to the case where N is a general c-net (with read arcs).

This result suggests that the reachability problem can be encoded as a satisfiability problem in propositional logic, using a formula whose size is polynomial in that of $\mathcal{U}_N$. Here, we shall only sketch the basics of how this can be done; future work will study the precise details needed to obtain practical model-checking algorithms.

The key idea is to first construct a formula ϕ_N that characterises the configurations and markings of $\mathcal{U}_N$. Thus, for every condition c and event e of $\mathcal{U}_N$, ϕ_N will contain a variable $\mathbf{c}$ and $\mathbf{e}$, respectively; the models of ϕ_N will be those assignments in which the event variables with value true correspond to some configuration $\mathcal{C}$ and the true condition variables to $\mathsf{Mark}(\mathcal{C})$.

We let $\phi_N := \phi_C \wedge \bigwedge_{c \in B} \phi_c$, where ϕ_C ensures the absence of asymmetric-conflict cycles in $\mathcal{C}$, and the formulae ϕ_c for $c \in B$ ensure causal closure of $\mathcal{C}$ and correct treatment of $\mathbf{c}$, i.e. variable $\mathbf{c}$ will be true iff condition c is produced but not consumed by $\mathcal{C}$.

A simple way to prohibit all cycles would be to list them explicitly, i.e.

$$\phi_C := \bigwedge_{e_1,\ldots,e_n \in \mathsf{Cycles}(\mathcal{U}_N)} \neg(e_1 \wedge \cdots \wedge e_n),$$

where $\mathsf{Cycles}(\mathcal{U}_N) := \{\, e_1, \ldots, e_n \mid e_1 \nearrow \cdots \nearrow e_n \nearrow e_1 \,\}$. Notice that this encoding is not polynomial as there may be exponentially many such cycles; however, better encodings exist with size $\mathcal{O}(n \cdot \log n)$, where n is the number of events [24].

Moreover, suppose that condition $c \in B$ has ${}^\bullet c = \{e\}$, $c^\bullet = \{f_1, \ldots, f_m\}$, and $\underline{c} = \{g_1, \ldots, g_k\}$. Then we define

$$\phi_c := \Big(\bigvee_{i=1}^{m} \mathbf{f}_i \vee \bigvee_{i=1}^{k} \mathbf{g}_i \Big) \rightarrow \mathbf{e} \quad \wedge \quad \mathbf{c} \leftrightarrow \Big(\mathbf{e} \wedge \bigwedge_{i=1}^{m} \neg \mathbf{f}_i \Big).$$

The main difference between ϕ_N and the corresponding construction for Petri nets in [1] is the treatment of asymmetric-conflict cycles. In Petri nets, all conflicts are symmetric and between pairs of events e, e' with ${}^\bullet e \cap {}^\bullet e' \neq \emptyset$. In contrast, conflict cycles in c-nets can be of arbitrary length as exemplified by Fig. 5, where e_1, e_2, e_3 form an asymmetric-conflict cycle of length 3.

Using ϕ_N, and following the example of [1], one can encode many questions about the set of reachable markings of N in terms of propositional logic, for instance:

- Is there any reachable marking in N that contains both places p and q?
  Let f is the mapping from Definition 1, and suppose that $c_1, \ldots, c_m \in B$ are those conditions with $f(c_i) = p$, for $i = 1, \ldots, m$, and $d_1, \ldots, d_k \in B$ those with $f(d_i) = q$, for $i = 1, \ldots, k$. Let $\phi_p = \mathbf{c}_1 \vee \cdots \vee \mathbf{c}_m$ and $\phi_q = \mathbf{d}_1 \vee \cdots \vee \mathbf{d}_k$. Then, a marking that contains both p and q exists in N iff
  $$\phi_N \wedge \phi_p \wedge \phi_q$$
  is satisfiable.

- Is $\{p, q\}$ a P-invariant of N?
  This means that every reachable marking puts exactly one token into either p or q. Under the same assumptions as above, this is the case iff
  $$\phi_N \rightarrow (\phi_p \oplus \phi_q)$$
  is valid, i.e., its negation is unsatisfiable.

- Does N contain a deadlock?
  Suppose that t is a transition of N with ${}^\bullet t = \{p, q\}$, and otherwise make the same assumptions as above. Let $\phi_t = \phi_p \wedge \phi_q$, then any model of $\phi_N \wedge \phi_t$ corresponds to a marking enabling t. Assuming that we construct corresponding formulae for all other transitions of N, then a deadlock exists iff
  $$\phi_N \wedge \bigwedge_{t \in T} \neg \phi_t$$
  is satisfiable.

9. Conclusions

We have made theoretical and practical contributions to the computation of unfoldings of contextual nets. To our knowledge, Cunf is the first tool that efficiently produces these objects. The availability of a tool that produces contextual unfoldings may trigger new interest in applications of c-nets and the algorithmics of asymmetric event structures in general.

It will be interesting to further explore the applications in verification, of which we have given a taste in Section 8. Unfolding-based techniques need two ingredients: an efficient method for generating them, and efficient methods for analysing the prefixes. We have provided the first ingredient in this quest. We believe that traditional unfolding-based verification techniques [25] can be extended to work with contextual unfoldings and that their succinctness may help to speed up these analyses. We find this topic to be an interesting avenue for future research.

Moreover, it would also be interesting to investigate a mix between eager and lazy that tries to get the best of the two worlds. For instance, one could start with the eager approach and switch (selectively for some conditions) to lazy as soon the number of compound conditions exceeds a certain bound. This, and other ideas, remain to be tested.

References

[1] J. Esparza, K. Heljanko, Unfoldings - A Partial-Order Approach to Model Checking, EATCS Monographs in Theoretical Computer Science, Springer, 2008.

[2] K. Heljanko, Combining symbolic and partial-order methods for model-checking 1-safe Petri nets, Ph.D. thesis, Helsinki University of Technology (2002).

[3] W. Vogler, A. L. Semenov, A. Yakovlev, Unfolding and finite prefix for nets with read arcs, in: Proc. of CONCUR'98, Vol. 1466 of LNCS, 1998, pp. 501–516.

[4] G. Ristori, Modelling systems with shared resources via Petri nets, Ph.D. thesis, Department of Computer Science, University of Pisa (1994).

[5] U. Montanari, F. Rossi, Contextual occurrence nets and concurrent constraint programming, in: Dagstuhl Seminar 9301, Vol. 776 of LNCS, 1994, pp. 280–295.

[6] R. Janicki, M. Koutny, Invariant semantics of nets with inhibitor arcs, in: Proc. Concur, Vol. 527 of LNCS, 1991, pp. 317–331.

[7] P. Baldan, A. Corradini, U. Montanari, An event structure semantics for P/T contextual nets: Asymmetric event structures, in: Proc. FoSSaCS, Vol. 1378 of LNCS, 1998, pp. 63–80.

[8] J. Winkowski, Reachability in contextual nets, Fundamenta Informaticae 51 (1–2) (2002) 235–250.

[9] P. Baldan, A. Corradini, B. König, S. Schwoon, McMillan's complete prefix for contextual nets, ToPNoC 1 (2008) 199–220, LNCS 5100.

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

[11] C. Rodríguez, Cunf, `http://www.lsv.ens-cachan.fr/~rodriguez/tools/cunf/`.

[12] S. Schwoon, Mole, `http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/`.

[13] P. Baldan, A. Bruni, A. Corradini, B. König, S. Schwoon, On the computation of McMillan's prefix for contextual nets and graph grammars, in: Proc. ICGT'10, Vol. 6372 of LNCS, 2010, pp. 91–106.

[14] C. Rodríguez, S. Schwoon, P. Baldan, Efficient contextual unfolding, in: Proc. Concur, Vol. 6901 of LNCS, 2011, pp. 342–357.

[15] C. Rodríguez, S. Schwoon, P. Baldan, Efficient contextual unfolding, Tech. Rep. LSV-11-14, LSV, ENS de Cachan (2011).

[16] R. Janicki, M. Koutny, Semantics of inhibitor nets., Information and Computation 123 (1995) 1–16.

[17] K. L. McMillan, Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits, in: Proc. CAV, Vol. 663 of LNCS, 1992, pp. 164–177.

[18] V. Khomenko, Model checking based on prefixes of Petri net unfoldings, Ph.D. thesis, School of Computing Science, Newcastle University (2003).

[19] V. Khomenko, Punf, `http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/`.

[20] K. Heljanko, Deadlock and reachability checking with finite complete prefixes, Licentiate's thesis, Helsinki University of Technology (1999).

[21] C. Rodríguez, Implementation of a complete prefix unfolder for contextual nets, Rapport de master, Master Parisien de Recherche en Informatique, Paris, France (Sep. 2010).

[22] K. Heljanko, Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets, Fundamenta Informaticae 37 (3) (1999) 247–268.

[23] C. Schröter, Halbordnungs- und Reduktionstechniken für die automatische Verifikation von verteilten Systemen, Ph.D. thesis, Universität Stuttgart (2006).

[24] M. Codish, S. Genaim, P. J. Stuckey, A declarative encoding of telecommunications feature subscription in sat, in: PPDP, 2009, pp. 255–266.

[25] J. Esparza, K. Heljanko, Implementing LTL model checking with net unfoldings, in: Proc. SPIN, Vol. 2057 of LNCS, 2001, pp. 37–56.