

Symbolic Unfoldings For Networks of Timed Automata

Franck Cassez¹, Thomas Chatain² and Claude Jard³

¹ CNRS/IRCCyN, Nantes, France – franck.cassez@cnrs.irccyn.fr

² IRISA/INRIA, Campus de Beaulieu, Rennes, France – Thomas.Chatain@irisa.fr

³ IRISA/ENS Cachan, Campus de Kerlann, Bruz, France – Claude.Jard@irisa.fr

Abstract In this paper we give a symbolic concurrent semantics for network of timed automata (NTA) in terms of symbolic nets. The aim is to keep explicit the notions of causality and concurrency among events. The result of the symbolic semantics is an infinite symbolic *unfolding*. We prove that there is a complete *finite* prefix for NTA, *i.e.* a finite prefix of the infinite unfolding, that already contains enough information to check many type of properties of the network, like reachability of a state. Moreover, we show that *extended nets* which are nets with *read arcs* can be used to build a complete finite prefix that contains accurate information w.r.t. to the firing dates of events.

1 Introduction

Concurrent Semantics for Finite State Systems. The analysis of *distributed* or *concurrent* finite state systems has been dramatically improved thanks to *partial-order* methods (see e.g. [30]) that take advantage of the *independence* between actions, and to the *unfolding* based methods [15,24] that improve the partial order methods by taking advantage of the *locality* of actions.

Timed Systems. The main models that include timing information and are used to specify distributed timed systems are networks of *timed* automata (NTA) [2,18], and *time* Petri nets (TPN) [25]. There are a number of theoretical results about NTA and TPN and efficient tools to analyze them have been developed ([3,10,21,19,8,17]). Nevertheless the analysis of these models is always based on the exploration of a graph which is a single large automaton that produces the same behaviours as the NTA or the TPN; this induces an exponential blow up in the size of the system to be analysed.

Related Work. In [20,26], the authors define an alternative semantics for NTA based on local time elapsing. The efficiency of this method depends on two opposite factors: local time semantics generate more states but the independence relation restricts the exploration. In [23] (a generalization of [31]), the independence between transitions in a TA is exploited in a different way: the key

observation is that the occurrences of two independent transitions do not need to be ordered and consequently nor do the occurrences of the clock resets. The relative drawback of the method is that, before their exploration, the symbolic states include more variables than the clock variables. Partial order methods for TPNs are studied in [28], where the authors generalize the concept of *stubborn set* to time Petri nets, calling it a *ready set*. They apply it to the *state class graph* construction of [7]. The efficiency of the method depends on whether the (dynamical) timing coupling between transitions is weak or not. Unfortunately the urgent semantics of this model entails a strong timing coupling. The previous *partial order* methods only take advantage of the independence of actions and not of any locality property. We are interested in a *true concurrent semantics* for NTA and this has not been developed in the aforementioned work.

Process semantics for time Petri nets which is a generalization of the unfolding semantics for time Petri nets has been developed by different researchers. From a semantical point of view, Aura and Lilius have studied in [27] the *realizability problem* of a non branching process in a TPN. They build an unfolding of the untimed Petri net underlying a safe TPN, and add *constraints* on the dates of occurrence of the events. It is then possible to check that a *timed configuration* is valid or not. In [16] the authors consider bounded TPN and a discrete time domain: the elapsing of one time unit is a special transition of the net. Thus the global synchronization related to this transition heavily decreases the locality property of the unfolding. Furthermore, when the intervals associated with the transitions involve large integers, this method suffers the usual combinatorial explosion related to the discrete time approach.

Section 3 of this paper can be viewed as the counterpart of the work of Aura and Lilius [27] in the framework of NTA: we define similar notions for NTA and build a *symbolic unfolding* which is a *symbolic net*. We have to extend the results of Aura and Lilius because there is no *urgency* for firing a transition¹ in a NTA. As stated in [27] those unfoldings are satisfactory for *free choice nets* which are a strict subclass of TPN. Our NTA are not free choice nets and in section 4 we refine our symbolic unfolding to obtain an *extended symbolic unfolding* which is a symbolic net with *read arcs*.

Following our recent approach using the notion of symbolic unfolding to capture the partial order behaviors of TPN [13], we propose in this paper a similar notion for NTA, but we cannot directly apply the framework of [13]. Indeed TA and TPN have different expressive powers ([6,12]) and as stated earlier NTA do not have the nice *urgency* features that TPN have.

Up to our knowledge, this is the first attempt to equip NTA with a concurrent semantics, which can be finitely represented by a prefix of an unfolding. In this paper we answer the following questions:

1. What can be a good model for a *concurrent semantics* of NTA ? The result is an extension of the model of symbolic nets we have proposed in [13];

¹ *invariants* and *guards* can be independent and a transition is not bound to fire before its deadline given by the *guard*.

2. How to define a *concurrent semantics* for NTA, *i.e.* how to define a *symbolic unfolding* that captures the essential properties of a NTA while preserving *concurrency information* ? This is achieved in two steps: first build a symbolic *(pre)-unfolding* and use this object to build a proper extended symbolic unfolding of the NTA. By “proper” unfolding, we mean a symbolic Petri net on which we can check that a *local configuration* is valid using only the *extended causal past* of an event.
3. Is there a *complete finite prefix* for NTA ? This result is rather easy to obtain on the *pre-unfolding* object and carries over to the symbolic unfolding.

About point 3 above, we are not addressing the problem of building such a prefix *efficiently* but our work is concerned with identifying the key issues in the construction of a prefix for NTA. The solution proposed in [13] builds a complete finite prefix for safe TPNs, but with no guarantee that this prefix is one of the smallest, which is a very difficult problem to solve. Based on this work, we address more basic questions about NTA, which are in a sense easier to study than safe TPNs because the concurrent structure is explicit.

Key Issues. In this section we present informally the problem and the key issues raised by the three previous questions. In the case of networks of finite automata, *finite complete prefixes* exist. For example, for the network of Fig. 1, a finite complete prefix² is given on Fig. 2. Finite complete prefixes contain full information about the reachable states of the network and about the set of events that are *feasible* in the network. A set of events (labels) is feasible if it can be generated by a *run* of the network.

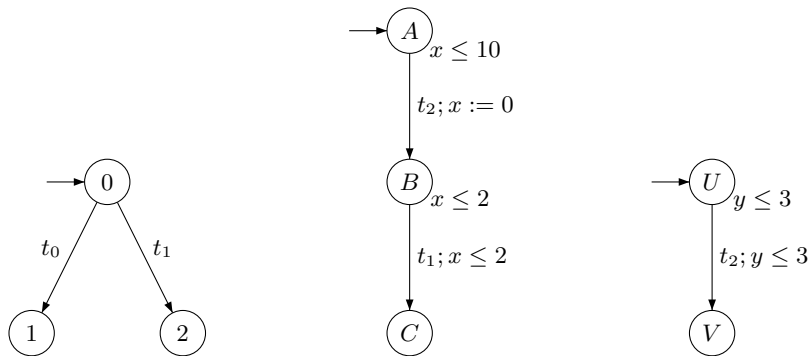


Figure 1. A network of three timed automata

² The automata synchronize on common labels. Labels of the events and places represent the corresponding location and transition in the network of automata. The constraints appearing near each node are explained later and can be ignored at this stage.

For example, $\{t_1\}$ is not a feasible set of events in the network of automata, because t_1 must be preceded by t_2 . And this appears in the unfolding. In an unfolding a set of events K is a *configuration* if there is a reachable marking obtained by firing each event in K . For example $\{\perp, e_1\}$ is a configuration, $\{\perp, e_1, e_2\}$ as well, but $\{\perp, e_3\}$ is not as e_3 must be preceded by e_2 before it occurs. The minimal set of events necessary for an event e to occur is called the *causal past* of e . Note that by definition a *configuration* contains the causal past of each of the event. A complete prefix has the property (P): a set of events is feasible in the network of automata iff it is a configuration of the unfolding³.

In the case of network of *timed automata*, we must use *timed events* which are pairs (e, δ) where $\delta \in \mathbb{R}_{\geq 0}$. To decide whether a set of timed events is feasible in a network of timed automata, we can think of building a *symbolic unfolding*. For this, we add a (*symbolic*) *timing constraint* $g(e)$ to each event of the previous unfolding. For example, with e_1 we can associate $g(e_1) \stackrel{def}{=} \delta_{e_1} - \delta_{\perp} \leq 5$, where δ_e is the variable that gives the date of occurrence of e . A set of timed events $\{(e_1, t_1), \dots, (e_k, t_k)\}$ is a *timed configuration* if $\{e_1, e_2, \dots, e_k\}$ is a configuration and the constraint $g(e_1) \wedge \dots \wedge g(e_k)$ is satisfied when replacing each δ_{e_i} by t_i . For example $\{(\perp, 0), (e_1, 4)\}$ is a timed configuration with $g(\perp) \stackrel{def}{=} \delta_{\perp} = 0$.

The property we would like to get for symbolic unfoldings is (P'): a set of timed events $\{(e_1, t_1), \dots, (e_k, t_k)\}$ is feasible iff it is a timed configuration.

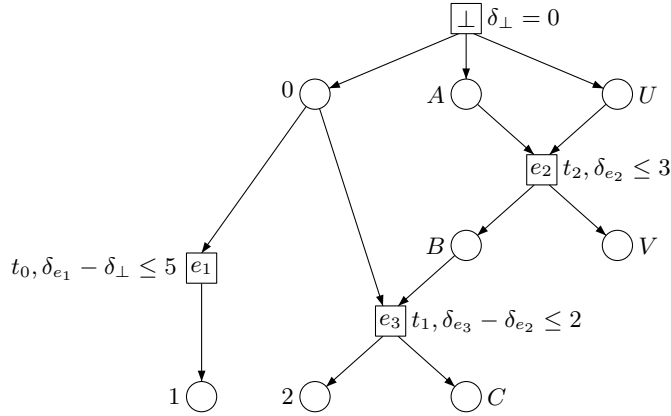


Figure 2. Symbolic unfolding for the example of Fig. 1

Note that a formula associated with an event e should only involve variables that are associated with events in the causal past of e . If this holds for each event, we say a symbolic unfolding is *consistent*. Indeed, if we set the constraint

³ Actually we should write “it is a labeling” of a configuration of the unfolding.

for e_1 to involve δ_{e_2} we cannot evaluate the conjunction of constraints $g(e_1) \wedge g(\perp)$. Now assume we want to decide whether $\{(\perp, 0), (e_1, t_1), (e_2, t_2)\}$ is a timed configuration. It is actually if $t_1 - t_2 \leq 2$. But this cannot be captured by any conjunction $g(\perp) \wedge g(e_1) \wedge g(e_2)$ because e_1 is not in the causal past of e_2 and e_2 not in the causal past of e_1 . A symbolic unfolding built by associating constraints to each event e , with the property that each constraint $g(e)$ uses only variables in the causal past of e , does not always contain enough information for property (P') to hold.

In this paper we address the following problems:

Problem 1 Given a network of timed automata, is there a symbolic finite complete prefix that contains full information about the reachable states ? We do require that the unfolding be consistent but allows to use constraints on the places as well as on events.

Problem 2 Is there a symbolic finite complete prefix that contains full information about the set of feasible *timed events* ? In this case, the unfolding should be consistent and contain only constraints on events.

Organization of the Paper. The paper is organized as follows. Section 2 presents the model of Networks of Timed Automata and its usual sequential semantics in term of product. Section 3 proposes a new concurrent semantics for NTA in terms of *symbolic branching processes* and proves the existence of complete finite prefixes. In section 4, we show how to transfer some timing constraints in the structure of the processes by using *extended processes*, basically branching processes with *read arcs*. Section 5 gives a summary of the paper and directions for future work.

2 Networks of timed automata

2.1 Notations.

Let $X = \{x_1, \dots, x_n\}$ be a finite set (clock variables). A *valuation* ν is a mapping from X to $\mathbb{R}_{\geq 0}$. Let $X' \subseteq X$. The valuation $\nu[X']$ is defined by: $\nu[X'](x) = 0$ if $x \in X'$ and $\nu[X'](x) = \nu(x)$ otherwise. $\nu|_{X'}$ is the restriction (projection) of ν to X' and is defined by $\nu|_{X'}(x) = \nu(x)$ for $x \in X'$. We denote $\mathbf{0}$ the valuation defined by $\mathbf{0}(x) = 0$ for each $x \in X$. For $\delta \in \mathbb{R}$, $\nu + \delta$ is the valuation defined by $(\nu + \delta)(x) = \nu(x) + \delta$. $\mathcal{C}(X)$ is defined to be the set of conjunctions of terms of the form $x - x' \bowtie c$ or $x \bowtie c$ for $x, x' \in X$ and $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. $\mathcal{C}(X)$ is called the set of *diagonal constraints* over X . The set of *rectangular constraints*, $\mathcal{C}_r(X)$ is the subset of $\mathcal{C}(X)$ where only constraints of the form $x \bowtie c$ appear. Given a formula $\varphi \in \mathcal{C}(X)$ and a valuation $\nu \in \mathbb{R}_{\geq 0}^X$, we denote $\varphi(\nu) \in \{\mathbf{tt}, \mathbf{ff}\}$ the truth value obtained by substituting each occurrence of x in φ by $\nu(x)$. We sometimes use $\varphi[x/\nu(x)]$ for φ where x is replaced by $\nu(x)$. We let $\llbracket \varphi \rrbracket = \{\nu \in \mathbb{R}_{\geq 0}^X \mid \varphi(\nu) = \mathbf{tt}\}$. A subset Z of $\mathbb{R}_{\geq 0}^X$ is a zone if $Z = \llbracket \varphi_Z \rrbracket$ for some $\varphi_Z \in \mathcal{C}(X)$. Note that the intersection of two zones is a zone. Two operators are defined on zones: the *time successor* operator, $Z' = \{v + \delta \mid v \in Z, \delta \in \mathbb{R}_{\geq 0}\}$ and

the R -reset operator, $Z[R] = \{v \mid \exists v' \in Z \text{ s.t. } v = v'[R]\}$. Both Z' and $Z[R]$ are zones if Z is a zone. A *simple substitution* s is a mapping $s : X \rightarrow \text{diff}(Y)$, where $\text{diff}(Y) = \{y - y' \mid y, y' \in Y\}$ that satisfies: if $s(x) = y - y'$ and $s(x') = z - z'$ then $z = y$. $\varphi[s]$ stands for the formula φ in which each occurrence of a variable $x \in X$ is replaced by $s(x)$. Thus if $\varphi \in \mathcal{C}(X)$ and s is a simple substitution, $\varphi[s] \in \mathcal{C}(Y)$.

2.2 Timed automata

Timed automata [2] are used to model systems which combine *discrete* and *continuous* evolutions.

Definition 1 (Timed Automaton). A timed automaton \mathcal{A} is a tuple $(L, \ell_0, \Sigma, X, T, \text{INV})$ where: L is a finite set of locations; ℓ_0 is the initial location; Σ is a finite set of discrete actions; $X = \{x_1, \dots, x_n\}$ is a finite set of (positive real-valued) clocks; $T \subseteq L \times \mathcal{C}_r(X) \times \Sigma \times 2^X \times L$ is a finite set of transitions: $(\ell, g, a, R, \ell') \in T$ represents a transition from the location ℓ to the location ℓ' , labeled by action a , with the guard g and the reset set $R \subseteq X$; we write $\text{SRC}(t) = \ell$, $\text{TGT}(t) = \ell'$, $\text{GUARD}(t) = g$, $\lambda(t) = a$ and $\text{RESET}(t) = R$. $\text{INV} \in \mathcal{C}_r(X)^L$ assigns an invariant to any location. We require that INV is downward closed i.e. a conjunctions of terms of the form $x \bowtie c$ with $\bowtie \in \{<, \leq\}$ and $c \in \mathbb{N}$.

A state of a timed automaton is a pair $(\ell, v) \in L \times \mathbb{R}_{\geq 0}^X$. A timed automaton is *bounded* if there exists a constant $k \in \mathbb{N}$ s.t. for each $\ell \in L$, $\text{INV}(\ell) \subseteq [0 \leq x_1 \leq k \wedge \dots \wedge 0 \leq x_n \leq k]$. Examples of timed automata are given in Fig. 1 and Fig. 4. The semantics of a timed automaton is a timed transition system.

Definition 2 (Semantics of a TA). The semantics of a timed automaton $\mathcal{A} = (L, \ell_0, \Sigma, X, T, \text{Inv})$ is a labeled timed transition system $S_{\mathcal{A}} = (Q, q_0, T \cup \mathbb{R}_{\geq 0}, \rightarrow)$ with $Q = L \times (\mathbb{R}_{\leq 0})^X$, $q_0 = (\ell_0, \mathbf{0})$ is the initial state and \rightarrow consists of the discrete and continuous transition relations:

- the discrete transition relation is defined for all $t \in T$ by:

$$(\ell, v) \xrightarrow{t} (\ell', v') \iff \exists t = (\ell, g, a, R, \ell') \in T \text{ s.t. } \begin{cases} g(v) = \text{tt}, \\ v' = v[R \mapsto 0] \\ \text{Inv}(\ell')(v') = \text{tt} \end{cases}$$

In the sequel we require that $g(v) \implies \text{INV}(\ell')(v[R])$ for each t ; this way the condition $\text{Inv}(\ell')(v') = \text{tt}$ can be omitted as it is always satisfied⁴ and we can decide whether $(\ell, v) \xrightarrow{t} (\ell', v')$ using only the source state (ℓ, v) and $\text{GUARD}(t)$.

⁴ Note that it does not restrict the model as for each transition $t = (\ell, g, a, R, \ell')$ we can compute a new guard g' s.t. replacing g by g' in t satisfies this property and the semantics of the automaton with g' is the same as with g .

– the continuous transition relation is defined for all $\delta \in \mathbb{R}_{\geq 0}$ by:

$$(\ell, v) \xrightarrow{\delta} (\ell', v') \iff \begin{cases} \ell = \ell' & v' = v + \delta \text{ and} \\ \forall 0 \leq \delta' \leq \delta, \text{Inv}(\ell)(v + \delta') = \text{tt} \end{cases}$$

A run of a timed automaton \mathcal{A} is a path in $S_{\mathcal{A}}$ starting in q_0 where continuous and discrete transitions alternate. Any run⁵ ρ can be written in the canonical form

$$\rho = q_0 \xrightarrow{\delta_0} q'_0 \xrightarrow{w_0} q_1 \xrightarrow{\delta_1} q'_1 \xrightarrow{w_1} q_2 \cdots q_n \xrightarrow{\delta_n} q'_n \xrightarrow{w_n} q_{n+1} \xrightarrow{\delta_{n+1}} q'_{n+1}$$

with $\delta_i \in \mathbb{R}_{\geq 0}$ and $w_i \in T$. The set of runs of \mathcal{A} is denoted by $[[\mathcal{A}]]$. A state q is reachable in \mathcal{A} if there is a run from q_0 to q . $\text{REACH}(\mathcal{A})$ is the set of reachable states of \mathcal{A} . The trace of a run is a timed word $(w_0, \delta_0)(w_1, \delta_1 + \delta_0) \cdots (w_n, \delta_n + \cdots + \delta_0)$ in $(T \times \mathbb{R}_{\geq 0})^*$. A timed word $w \in (T \times \mathbb{R}_{\geq 0})^*$ is accepted by \mathcal{A} if there is a run $\rho \in [[\mathcal{A}]]$ s.t. the trace of ρ is w . $\mathcal{L}(\mathcal{A})$ is the set of timed word accepted by \mathcal{A} .

The analysis [1,5,9,29,11] of timed automata is based on the exploration of a (finite) graph, the *simulation graph*, where the nodes are *symbolic states*. A symbolic state is a pair (ℓ, Z) where ℓ is a location and Z a zone over the set $\mathbb{R}_{\geq 0}^X$.

Definition 3 (Simulation Graph). The simulation graph $SG(\mathcal{A})$ of a timed automaton \mathcal{A} is given by:

- the set of states is the set of symbolic states of the form (ℓ, Z) where Z is a zone;
- the initial state is (ℓ_0, Z_0) with $Z_0 = \mathbf{0}^{\nearrow} \cap [[\text{INV}(\ell_0)]]$;
- $(\ell, Z) \xrightarrow{a} (\ell', Z')$ if there is a transition (ℓ, g, a, R, ℓ') in \mathcal{A} s.t. $Z \cap [[g]] \neq \emptyset$ (this ensures Z' is not empty) and $Z' = ((Z \cap [[g]])[R])^{\nearrow} \cap [[\text{INV}(\ell')]]$.

We assume that the timed automata are bounded *i.e.* in each location ℓ , $\text{Inv}(\ell)$ is bounded⁶. In this case the number of zones of the simulation graph is finite [22,9]. As the name indicates, the simulation graph simulates (in a time abstract way) the timed automaton. It preserves the untimed language of the timed automaton and contains only symbolic states that are reachable.

2.3 Product of Timed Automata.

It is convenient to describe a system as a parallel composition of timed automata. To this end, we use the classical composition notion based on a *synchronization function* à la Arnold-Nivat. Let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be n timed automata with $\mathcal{A}_i =$

⁵ In our definition runs are labeled by transitions.

⁶ Any timed automaton can be transformed into an equivalent (behaviours) bounded automaton [4].

$(L_i, l_{i,0}, \Sigma_i, X_i, T_i, \text{Inv}_i)$. We assume that for each $i \neq j$, $L_i \cap L_j = \emptyset$ and $X_i \cap X_j = \emptyset$. Given a set B we use B^ε for the set $B \cup \{\varepsilon\}$ (assuming $\varepsilon \notin B$).

A *synchronization constraint* I is a subset of $\Sigma_1^\varepsilon \times \Sigma_2^\varepsilon \cdots \times \Sigma_n^\varepsilon \setminus (\varepsilon, \dots, \varepsilon)$. The (synchronization) vectors of a synchronization constraint I indicate which actions synchronize. If $z = (a_1, \dots, a_n) \in I$ we write $z[j]$ for the j -th component a_j . For $(t_1, \dots, t_n) \in T_1^\varepsilon \times \cdots \times T_n^\varepsilon$ we write $\lambda(t_1, \dots, t_n) = (\lambda_1(t_1), \dots, \lambda_n(t_n))$ with $\lambda_i(\varepsilon) = \varepsilon$. For $z \in I$, we define $\lambda^{-1}(z) = \{t \in T_1^\varepsilon \times \cdots \times T_n^\varepsilon \mid \lambda(t) = z\}$ and $\lambda^{-1}(I)$ to be the union of the sets of $\lambda^{-1}(z)$ for $z \in I$. $\lambda^{-1}(I)$ indicates how the transitions synchronize. For $t \in \lambda^{-1}(I)$, we let: $\text{SRC}^*(t) = \{l \in \text{SRC}(t[i]) \mid t[i] \neq \varepsilon\}$, $\text{TGT}^*(t) = \{l \in \text{TGT}(t[i]) \mid t[i] \neq \varepsilon\}$, $\text{RESET}(t) = \{x \mid x \in \text{RESET}(t[i]) \text{ and } t[i] \neq \varepsilon\}$, $\text{GUARD}(t) = \bigwedge_{t[i] \neq \varepsilon} \text{GUARD}(t[i])$.

Definition 4 (Synchronous Product of Timed Automata). *The synchronous product $(\mathcal{A}_1 \mid \dots \mid \mathcal{A}_n)_I$ is the timed automaton $\mathcal{B} = (L, \mathbf{l}_0, \Sigma, X, T, \text{INV})$ defined by: $L = L_1 \times \cdots \times L_n$, $\mathbf{l}_0 = (\ell_{1,0}, \dots, \ell_{n,0})$, $\Sigma = \Sigma_1 \times \cdots \times \Sigma_n$, $X = \cup_{i=1}^n X_i$; $(\mathbf{l}, g, a, R, \mathbf{l}') \in T$ iff $\exists t \in \lambda^{-1}(I)$ s.t.: (1) if $t[i] \neq \varepsilon$ then $\mathbf{l}_i = \text{SRC}(t[i])$ and otherwise $\mathbf{l}'_i = \text{TGT}(t[i])$, (2) $a = \lambda(t)$, $g = \text{GUARD}(t)$ and $R = \text{RESET}(t)$. The invariant mapping is given by $\text{INV}(\mathbf{l}) = \bigwedge_{i=1}^n \text{INV}_i(\ell_i)$ if $\mathbf{l} = (\ell_1, \dots, \ell_n)$.*

This definition implies that if each \mathcal{A}_i is bounded then the product is bounded. Also if $\text{GUARD}(t_i)(v_i) \implies \text{INV}(\ell'_i)(v_i[R_i])$ for each $t_i = (\ell_i, g_i, a_i, R_i, \ell'_i)$ then $\text{GUARD}(t)(v) \implies \text{INV}(\text{TGT}(t))(v[\text{RESET}(t)])$ in the product (clocks are not shared).

3 Symbolic Unfolding for Network of Timed Automata

In this section we define the symbolic semantics of a network of TA in terms of *symbolic branching processes*. Those processes contain timing constraints both on places and events.

3.1 Symbolic Occurrence Nets

Definition 5 (Net). *A net \mathcal{N} is a tuple $(E, P, \bullet(), ()^\bullet)$ where E is a set of events, P a set of places, and $\bullet()$ and $()^\bullet$ are two mappings defined on $E \cup P$ s.t. if $e \in E$, $\bullet e \subseteq P$ and $e^\bullet \subseteq P$ and if $p \in P$, $\bullet p \subseteq E$ and $p^\bullet \subseteq E$.*

We assume that each net has a special event $\perp \in E$, s.t. $\bullet \perp = \emptyset$ and for any other $x \in E \cup P$, $\bullet x \neq \emptyset$. \perp is the *minimal element* of the net. The semantics of a net is the usual one for Petri nets: we assume that the minimal event \perp fires only once (and puts tokens in an *initial marking*). We refer to *nodes* of a net for any element of $E \cup P$. When it is convenient we use the mappings $\bullet()$ and $()^\bullet$ on sets of nodes with the obvious meaning.

Let x, y be two nodes. If $x \in \bullet y$ or $y \in x^\bullet$ there is an *arc* from x to y and we write $x \rightarrow y$. This enables us to refer to the *directed graph of a net* which is simply the graph $(E \cup P, \rightarrow)$. The reflexive and transitive closure of \rightarrow is

denoted \preceq . x, y are *causally related* if either $x \preceq y$ or $y \preceq x$. x is in the (strict) *causal past* of y if $x \preceq y$ and $x \neq y$, i.e. $x \prec y$. x, y are in *conflict*, noted $x \# y$, if there is a place $p \in P$ such that $p \rightarrow w \preceq x$ and $p \rightarrow u \preceq y$ with $u \neq w$. x and y are *concurrent* if x and y are neither causally related nor in conflict. If J is a set of events then $\uparrow J = (\cup_{e \in J} e^\bullet) \setminus (\cup_{e \in J} \bullet e)$. For a set $J \subseteq E \cup P$ $\lceil J \rceil = \{e' \in E \cup P \mid e' \preceq e \text{ for some } e \in J\}$. A set of events J is *causally closed* if $\lceil J \rceil = J$. A set A is a *co-set* if any two elements of A are concurrent.

Definition 6 (Occurrence Net). An occurrence net $\mathcal{O} = (E, P, \bullet(), ()^\bullet)$ is a net satisfying the following properties:

- the directed graph $(E \cup P, \rightarrow)$ has no cycles,
- for each $p \in P$, $\bullet p$ contains at most one element,
- no node is in self-conflict, i.e. there is no $x \in E \cup P$ s.t. $x \# x$.

A *configuration* of an occurrence net $\mathcal{O} = (E, P, \bullet(), ()^\bullet)$ is a set of events $K \subseteq E$ which is causally closed and conflict-free. A *cut* $S \subseteq P$ is a set of places which is a maximal co-set. To each configuration K , we can associate a unique cut $\uparrow K$ which is denoted $\text{CUT}(K)$.

Given a set B we denote $\delta(B)$ the set of (fresh) variables $\{\delta_b \mid b \in B\}$.

Definition 7 (Symbolic Occurrence Nets). A symbolic occurrence net \mathcal{T} is a tuple $(E, P, \bullet(), ()^\bullet, \gamma)$ where $(E, P, \bullet(), ()^\bullet)$ is an occurrence net, and $\gamma : E \cup P \rightarrow \mathcal{C}(X)$ with $X = \delta(E \cup P)$. We require that:

1. for each $x \in E \cup P$, $\gamma(x)$ contains only variables in $\delta(\lceil x \rceil)$,
2. $\gamma(\perp) \stackrel{\text{def}}{=} (\delta_\perp = 0)$.

We refer to the net $(E, P, \bullet(), ()^\bullet)$ as the underlying net of \mathcal{T} .

An example of a symbolic net is given in Fig. 2, 4: places are given as circles and the constraint γ appears on one side. Events are in boxes with their constraints on the side as well.

We now define the *symbolic cuts* of an symbolic net.

Definition 8 (Symbolic Cut). (M, Φ) is a symbolic cut of \mathcal{T} if:

1. M is a cut of the underlying net, i.e. there is some configuration K of underlying net s.t. $M = \text{CUT}(K)$,
2. $\Phi = \Phi_1(M) \wedge \Phi_2(M) \wedge \Phi_3(M) \wedge \Phi_4(M)$ where $\Phi_i(M), 1 \leq i \leq 4$ are defined by:

$$\Phi_1(M) = \bigwedge_{x \in \lceil M \rceil} \gamma(x) \quad (1)$$

$$\Phi_2(M) = \bigwedge_{e \in \lceil M \rceil \cap E} (\bigwedge_{p \in \bullet e} \delta_p = \delta_e) \quad (2)$$

$$\Phi_3(M) = \bigwedge_{p \in M} (\delta_{\bullet p} \leq \delta_p) \quad (3)$$

$$\Phi_4(M) = (\bigwedge_{p, p' \in M} \delta_p = \delta_{p'}) \quad (4)$$

Notice that the formula Φ of a symbolic cut is entirely determined by the cut M and unique; we denote it by Φ_M . In the sequel we will use Φ_C when C is a co-set instead of a cut; definition 8 then rewrites as: 1) C is a co-set of the underlying net and 2) is unchanged using $\lceil C \rceil$ instead of $\lceil M \rceil$. Let M be a cut $\nu : \delta(\lceil M \rceil) \rightarrow \mathbb{R}_{\geq 0}$. (M, ν) is a *timed cut* if $\nu \in \llbracket \Phi_M \rrbracket$. Consequently $(M, \llbracket \Phi_M \rrbracket)$ is the set of timed cuts associated with M .

3.2 Symbolic Processes for Network of Timed Automata

Let $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ be a synchronous product of TA. We assume the set of locations of the \mathcal{A}_i are 2 by 2 disjoint and let $L = \cup_{1 \leq i \leq n} L_i$. The *symbolic branching processes* (SBPs) of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ are symbolic occurrence nets built over two sets \mathcal{E} and \mathcal{P} defined inductively by:

- $\perp \in \mathcal{E}$,
- if $e \in \mathcal{E}$ and $s \in L$ then $(e, s) \in \mathcal{P}$,
- if $S \subseteq \mathcal{P}$ and $t \in I$ then $(S, t) \in \mathcal{E}$.

On those two sets we define the mappings $\bullet()$, $()^\bullet$ by

- for \mathcal{E} : $\bullet \perp = \emptyset$, and if $e = (S, t)$, $\bullet e = S$ and $e^\bullet = \{s \mid (e, s) \in \mathcal{P}\}$;
- for \mathcal{P} : $\bullet(e, s) = e$ and $(e, s)^\bullet = \{e \mid \bullet e \cap s \neq \emptyset\}$.

By definition of E and P a SBP is completely determined by E and P as $\bullet \cdot$ and \cdot^\bullet are implicitly defined. (E, P, γ) with $E \subseteq \mathcal{E}$ and $P \subseteq \mathcal{P}$, is a SBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ if γ satisfies conditions (1–3) of Def. 7. We use the following notations:

- for $e = (S, t) \in \mathcal{E} \setminus \{\perp\}$, $\lambda(e) = t$; we say that e is an i -event if $t[i] \neq \varepsilon$;
- for $(e, s) \in \mathcal{P}$, $\text{loc}(e, s) = s$; we let $\text{Loc}(Y) = \cup_{y \in Y} \text{loc}(y)$. (e, s) is an i -place if $s \in L_i$;
- $\text{RESET}(e) = \text{RESET}(\lambda(e))$, and by convention $\text{RESET}(\perp) = X$.
- if $x \in X$ and K is a configuration of a SBP, the *event of last reset of x in K* is $Z_K^{\neq 0}(x) = \min\{e \in K \mid x \in \text{RESET}(e)\}$. A minimal element always exist as \perp resets each variable. The unicity of this element will follow from the fact that each variable belongs to one automaton and that the events of each automaton are totally ordered.

Definition 9 (Symbolic Finite Branching Processes). *The set of symbolic finite branching processes of a network $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ is defined inductively as follows:*

- $(\{\perp\}, \{(\perp, l_{1,0}), (\perp, l_{2,0}), \dots, (\perp, l_{n,0})\}, \gamma)$ with $\gamma(\perp) \stackrel{\text{def}}{=} \delta_\perp = 0$, $\gamma(\perp, l_{i,0}) \stackrel{\text{def}}{=} \text{INV}(l_{i,0})[s_i]$, with the simple substitution s_i defined by $s_i(x) = \delta_{(\perp, l_{i,0})}$ for each $x \in X_i$, is a SBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$;
- if (E, P, γ) is a SBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$, $t \in I$, $C \subseteq P$ is a co-set s.t. $\text{Loc}(C) = \text{SRC}^*(t)$, then

$$(E \cup \{e\}, P \cup \{(e, s) \mid s \in \text{TGT}^*(t)\}, \gamma')$$

is a SBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$, with $e = (C, t)$, $\gamma'_{E \cup P} = \gamma$, $\gamma(e) = \text{GUARD}(t)[\eta]$, $\eta : x \mapsto \delta_e - \delta_{Z_{\lceil C \rceil}^{\neq 0}}(x)$, $\gamma(e, s) = \text{Inv}(s)[\eta']$, $\eta' : x \mapsto \delta_{(e, s)} - \delta_{Z_{\lceil e \rceil}^{\neq 0}}(x)$ (note that η and η' are simple substitutions.)

If $e \notin E$, e is a possible extension of (E, P) .

A SBP of a network is completely determined by E and P as the mapping γ is completely determined by E and P . By definition of the SBP, each symbolic cut (M, Φ_M) is such that $\Phi_M \in \mathcal{C}(\delta(\lceil M \rceil))$. By construction, a symbolic branching process satisfies conditions (1–3) of Definition 7. An example of a symbolic branching process is given on Fig. 2. The label on the side of a place p is $\text{loc}(p)$. The constraint $\gamma(p)$ appears on the side. For an event, the label $\lambda(e)$ appears on the side along with the constraint $\gamma(e)$.

We define the union of two symbolic branching processes (E_1, P_1, γ_1) and (E_2, P_2, γ_2) component-wise on events and places $(E_1 \cup E_2, P_1 \cup P_2, \gamma)$, and⁷ $\gamma(x) = \gamma_i(x)$ if $x \in E_i \cup P_i$. We also accept as symbolic branching processes countable unions of finite branching processes, which are infinite symbolic branching processes. Then symbolic branching processes are closed under countable union and we can define the *symbolic unfolding*, $\text{TBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$, of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ to be the maximal symbolic branching process.

Our SBPs are simple extensions of *branching processes* as defined in [14]. The discrete structure of a SBP is a BP. Also the constraints on the nodes can only *restrict* the reachable marking of the BP underlying a SBP. Hence the next two properties are easy to prove, because they already hold for untimed network ([14]).

Proposition 1. *Two i -nodes of a SBP are either causally related or in conflict.*

Proposition 2. *Let c be a cut of a SBP. Then c contains exactly one i -place for each $1 \leq i \leq n$.*

This enables us to associate with each configuration K , a symbolic state. Let $\text{CUT}(K)$ be the cut associated with K . Because of Proposition 2 above, we can associate a unique discrete state \mathbf{l} of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ with $\text{CUT}(K)$: \mathbf{l} is the “vector” representation of the set $\text{Loc}(\text{CUT}(K))$. Now consider the formula $f = \Phi_{\text{CUT}(K)}$ defined in Definition 8. This formula is in $\mathcal{C}(\delta(\lceil \text{CUT}(K) \rceil))$. Assume $\nu \in \llbracket f \rrbracket$. Define $v_\nu \in \mathbb{R}_{\geq 0}^X$ by⁸: $v_\nu(x) = \nu(\text{CUT}(K)) - \nu(\delta_{Z_{\lceil K \rceil}^{\neq 0}}(x))$. We let $Z_K = \{v_\nu \mid \nu \in \llbracket f \rrbracket\}$. The symbolic global state associated with K is $\text{GS}(K) = (\text{Loc}(\text{CUT}(K)), Z_K)$. Note that Z_K can be the empty set.

We now prove that $\text{TBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ contains correct and complete information w.r.t. the simulation graph of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$:

Theorem 1 (Weak Correctness). *If K is a configuration s.t. $\text{GS}(K) \neq \emptyset$ then*

1. $\text{GS}(K) = (\mathbf{l}, Z)$ for some (\mathbf{l}, Z) reachable in $\text{SG}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$,

⁷ As γ is uniquely defined for any $x \in E_i \cup P_i$, if $x \in E_1 \cap E_2$ we must have $\gamma_1(x) = \gamma_2(x)$; the same holds for $P_1 \cup P_2$.

⁸ Equation (4) implies that for each $p, p' \in \text{CUT}(K)$, $\nu(\delta_p) = \nu(\delta_{p'})$ and we denote this value $\nu(\text{CUT}(K))$.

2. if $K \cup \{e\}$ is a configuration and $\llbracket \Phi_{\text{CUT}(K)} \wedge \gamma(e) \wedge (\bigwedge_{p \in \bullet_e} \delta_p = \delta_e) \rrbracket \neq \emptyset$ then $(\mathbf{1}, Z) \xrightarrow{\lambda(e)} (\mathbf{1}', Z')$ and $\text{GS}(K \cup \{e\}) = (\mathbf{1}', Z')$.

Before going into the proof we point out an important thing: the formula $\Phi_{\text{CUT}(K)} \wedge \gamma(e) \wedge (\bigwedge_{p \in \bullet_e} \delta_p = \delta_e)$ is actually a formula over $\delta(K \cup \{e\})$ because if $\Phi = \Phi_{\text{CUT}(K)}$, Φ_2 implies that each $\delta_p, p \in P \cap \lceil K \rceil$ is equal to some $\delta_e, e \in E \cap \lceil K \rceil$; and because of the formula $\bigwedge (\bigwedge_{p \in \bullet_e} \delta_p = \delta_e)$ and Φ_4 , for each $p \in \text{CUT}(K)$ we must have $\delta_p = \delta_e$.

Proof. We prove that: if K is a cut and $\text{GS}(K) = (\mathbf{1}, Z)$ for some $(\mathbf{1}, Z)$ reachable in $SG((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$, then: if $K \cup \{e\}$ is a configuration and $\llbracket \Phi_{\text{CUT}(K)} \wedge \gamma(e) \wedge (\bigwedge_{p \in \bullet_e} \delta_p = \delta_e) \rrbracket \neq \emptyset$ then $(\mathbf{1}, Z) \xrightarrow{\lambda(e)} (\mathbf{1}', Z')$ and $\text{GS}(K \cup \{e\}) = (\mathbf{1}', Z')$. This together with $\text{GS}(\{\perp\}) = (\mathbf{1}_0, Z_0)$ implies Theorem 1.

Let $\nu \in \llbracket \Phi_{\text{CUT}(K)} \wedge \gamma(e) \wedge (\bigwedge_{p \in \bullet_e} \delta_p = \delta_e) \rrbracket$. As $\Phi_2(\text{CUT}(K))$ and $\Phi_3(\text{CUT}(K))$ holds for ν , together with $(\bigwedge_{p \in \bullet_e} \delta_p = \delta_e)$ this implies that $\nu(\delta_e) \geq \nu(\delta_{e'})$ for each $e' \in K$. Now extend ν to e^\bullet by $\nu(\delta_p) = \nu(\delta_e)$ for $p \in e^\bullet$. As $\Phi_4(\text{CUT}(K))(\nu)$ holds and because of the definition of the extension of ν to e^\bullet , $\Phi_4(\text{CUT}(K \cup \{e\}))(\nu)$ holds. $\Phi_3(\text{CUT}(K \cup \{e\}))$ and $\Phi_2(\text{CUT}(K \cup \{e\}))$ hold as well because $(\bigwedge_{p \in \bullet_e} \delta_p = \delta_e)(\nu)$ holds. It remains to prove that $\gamma(p)(\nu)$ holds for $p \in e^\bullet$. We know that $\gamma(e)(\nu)$ holds. Let v be the valuation defined by $v(x) = \nu(\delta_e) - \nu(\delta_{Z_{\overline{K}} := 0}(x))$. As $\text{GS}(K) = (\mathbf{1}, Z)$, this implies that $v \in Z$. Using the definition of $\gamma(e)$ we obtain that $\text{GUARD}(t)(v)$ holds. Because $v \in Z \cap \llbracket \text{GUARD}(t) \rrbracket$ the invariant in the state that is the target of t holds, *i.e.*, $\text{INV}(\text{loc}(p))(v[\text{RESET}(t)])$ holds for each $p \in e^\bullet$. This in turn implies that $\text{INV}(\text{loc}(p))(\nu)$ holds and thus $\gamma(p)(\nu)$ holds for each $p \in e^\bullet$. Hence $\Phi_1(\text{CUT}(K \cup \{e\}))(\nu)$ holds. Note that $v \in Z \cap \llbracket \text{GUARD}(t) \rrbracket$ also implies that $(\mathbf{1}, Z) \xrightarrow{\lambda(e)} (\mathbf{1}', Z')$.

Let $K' = K \cup \{e\}$. We know that K' is a configuration and $\text{GS}(K') \neq \emptyset$. Let $\nu' \in \llbracket \Phi_{\text{CUT}(K')} \rrbracket$. There must be an event $e' \in K'$ s.t. $\nu'(\delta_{e'}) \geq \nu'(\delta_{e''})$. Without loss of generality and to simplify the proof, assume event e enjoys this property: $\nu'(\delta_e) \geq \nu'(\delta_{e''})$ for each $e'' \in K$. By definition $K = K' \setminus \{e\}$. Define ν'' on $\lceil \text{CUT}(K') \rceil$ by: $\nu''(\delta_x) = \nu'(\delta_x)$ for $x \in \lceil K' \rceil$ and $\nu''(\delta_x) = \nu(\delta_e)$ for $x \in \text{CUT}(K')$. ν'' satisfies $\Phi_{\text{CUT}(K')}$ as well. Assume $\Phi_1'', \Phi_2'', \Phi_3''$ and Φ_4'' denotes the different parts of $\Phi_{\text{CUT}(K')}$ as defined by equations (1)–(4). ν'' satisfies Φ_4'' by definition of ν'' . ν'' satisfies Φ_3'' : for each $p \in \lceil K' \rceil$, $\nu''(\delta_{\bullet_p}) \leq \nu''(\delta_p)$ because ν'' agrees with ν' on $\lceil K' \rceil$; let $p \in \text{CUT}(K')$; as $\nu(\delta_e) \geq \nu(\delta_{e'})$ for each $e' \in K'$, it entails $\nu''(\delta_{\bullet_p}) \leq \nu''(\delta_p)$ as well. Φ_2'' is satisfied as well by ν'' : it involves only nodes in $\lceil K' \rceil$ and ν' and ν'' agrees on $\lceil K' \rceil$. We have now to check that Φ_1'' holds for ν'' . For $x \in \lceil K' \rceil$ $\gamma(x)(\nu')$ holds because again ν' and ν'' agrees on $\lceil K' \rceil$. Let $p \in \text{CUT}(K')$. We need to check that $\text{INV}(\text{loc}(p))[x \mapsto \delta_p - \delta_{Z_{\overline{K'}} := 0}(x)]$ holds for ν'' . By assumption we know it holds for ν' . Hence $\text{INV}(\text{loc}(p))[x / (\nu'(\text{CUT}(K)) - \nu'(\delta_{Z_{\overline{K'}} := 0}(x)))]$ holds. $\nu(\delta_e) \leq \nu(\delta_p)$ for $p \in \text{CUT}(K')$ and $\nu''(\delta_{Z_{\overline{K'}} := 0}(x)) = \nu'(\delta_{Z_{\overline{K'}} := 0}(x))$. As invariants are downward closed, this entails $\text{INV}((\text{loc}(p))[x / (\nu''(\delta_e) - \nu''(\delta_{Z_{\overline{K'}} := 0}(x)))]$ holds. And this implies $\gamma(p)(\nu'')$ holds. Hence Φ_1'' holds for ν'' and $\nu'' \in \llbracket \Phi_{\text{CUT}(K')} \rrbracket$.

We now define ν on $\lceil \text{CUT}(K) \rceil$ by: $\nu = \nu''_{\lceil \text{CUT}(K) \rceil}$ and prove that $\nu \in \llbracket \Phi_{\text{CUT}(K)} \rrbracket$. Assume $\Phi_{\text{CUT}(K)} = \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \Phi_4$ with Φ_i defined as in equations (1)–(4). We need to prove that $\Phi_i(\nu)$ holds for $1 \leq i \leq 4$. $\Phi_2(\nu)$ holds because ν and ν'' agrees on $\lceil K \rceil$. $\Phi_3(\nu)$ holds for the same reason. For Φ_4 , let $p \in \text{CUT}(K)$. Either $p \in \text{CUT}(K')$ or $p \in \bullet e$. In the first case, $\nu(\delta_p) = \nu''(\delta_e)$. In the latter case ($p \in \bullet e$), $\nu(\delta_p) = \nu''(\delta_e)$ as well. For Φ_1 , $\gamma(x)(\nu)$ holds for $x \in \lceil K \rceil$ because ν and ν'' agrees on $\lceil K \rceil$. Let $p \in \text{CUT}(K)$. If $p \in \text{CUT}(K')$, $\gamma(p)(\nu)$ holds because $\gamma(x)(\nu'')$ holds. Otherwise, $p \in \bullet e$. In this case because $\Phi_2(\nu)$ holds and $\nu(\delta_p) = \nu(\delta_e) = \nu''(\delta_p)$ it follows that $\gamma(p)(\nu)$ holds. This proves that $\nu \in \llbracket \Phi_{\text{CUT}(K)} \rrbracket$ and thus, K is a configuration and $\text{GS}(K) \neq \emptyset$.

The following part of the proof will be re-used for the proof of Theorem 2. We now prove that $v' \in \text{GS}(K \cup \{e\})$ implies $v' \in Z'$. Let $\nu' \in \llbracket \Phi_{\text{CUT}(K \cup \{e\})} \rrbracket$ and $v'(x) = \nu'(\text{CUT}(K \cup \{e\})) - \nu'(\delta_{Z_{K \cup \{e\}}^0(x)})$.

There is a finite number of events in K' and we take some event e s.t. $\nu'(\delta_e) \geq \nu'(\delta_{e'})$ for each $e' \in K'$. We let $K = K' \setminus \{e\}$. Define ν'' on $\lceil \text{CUT}(K') \rceil$ by: $\nu''(\delta_x) = \nu'(\delta_x)$ for $x \in \lceil K' \rceil$ and $\nu''(\delta_x) = \nu(\delta_e)$ for $x \in \text{CUT}(K')$. ν'' satisfies $\Phi_{\text{CUT}(K')}$ as well. Assume $\Phi''_1, \Phi''_2, \Phi''_3$ and Φ''_4 denotes the different parts of $\Phi_{\text{CUT}(K')}$ as defined by equations (1)–(4). ν'' satisfies Φ''_4 by definition of ν'' . ν'' satisfies Φ''_3 : for each $p \in \lceil K' \rceil$, $\nu''(\delta_{\bullet p}) \leq \nu''(\delta_p)$ because ν'' agrees with ν' on $\lceil K' \rceil$; let $p \in \text{CUT}(K')$; as $\nu(\delta_e) \geq \nu(\delta_{e'})$ for each $e' \in K'$, it entails $\nu''(\delta_{\bullet p}) \leq \nu''(\delta_p)$ as well. Φ''_2 is satisfied as well by ν'' : it involves only nodes in $\lceil K' \rceil$ and ν' and ν'' agrees on $\lceil K' \rceil$. We have now to check that Φ''_1 holds for ν'' . For $x \in \lceil K' \rceil$ $\gamma(x)(\nu')$ holds because again ν' and ν'' agrees on $\lceil K' \rceil$. Let $p \in \text{CUT}(K')$. We need to check that $\text{INV}(\text{loc}(p))[x \mapsto \delta_p - \delta_{Z_{K'}^0(x)}]$ holds for ν'' . By assumption we know it holds for ν' . Hence $\text{INV}(\text{loc}(p))[x / (\nu'(\text{CUT}(K)) - \nu'(\delta_{Z_{K'}^0(x)}))]$ holds. $\nu(\delta_e) \leq \nu(\delta_p)$ for $p \in \text{CUT}(K')$ and $\nu''(\delta_{Z_{K'}^0(x)}) = \nu'(\delta_{Z_{K'}^0(x)})$. As invariants are downward closed, this entails $\text{INV}(\text{loc}(p))[x / (\nu''(\delta_e) - \nu''(\delta_{Z_{K'}^0(x)}))]$ holds. And this implies $\gamma(p)(\nu'')$ holds. Hence Φ''_1 holds for ν'' and $\nu'' \in \llbracket \Phi_{\text{CUT}(K')} \rrbracket$.

We build the valuations ν'' and ν as before. The valuations ν'' and ν associated with ν'' and ν satisfy: $v' = \nu'' + (\nu'(\text{CUT}(K)) - \nu'(\delta_e))$ and $v'' = \nu[\text{RESET}(\lambda(e))]$. Also v' satisfies $\text{INV}(\mathbf{I})$ (using the definitions of $\gamma(x)$ and the fact that $\gamma(x)$ holds for ν' for $x \in \text{CUT}(K')$). This proves that $v' \in Z'$. Now let $v' \in Z'$. We need to prove that v' can be obtained from some $\nu' \in \llbracket \Phi_{\text{CUT}(K')} \rrbracket$. By definition of Z' there exists ν'' , v and $\delta \geq 0$ s.t. $v' = \nu'' + \delta$ and $\nu'' = \nu[\text{RESET}(\lambda(e))]$ for some $\nu \in Z \cap \llbracket \text{GUARD}(\lambda(e)) \rrbracket$ and $\nu' \in \llbracket \text{INV}(\mathbf{I}) \rrbracket$. As $\nu \in Z \cap \llbracket \text{GUARD}(\lambda(e)) \rrbracket$, $\nu \in \llbracket \Phi_{\text{CUT}(K)} \rrbracket$ (because $\text{GS}(K) = (\mathbf{1}, Z)$). Hence there exists some $\nu \in \llbracket \Phi_{\text{CUT}(K)} \rrbracket$ s.t. $\nu(x) = \nu(\text{CUT}(K)) - \nu(\delta_{Z_K^0(x)})$. Also, $\nu \in \llbracket \text{GUARD}(\lambda(e)) \rrbracket$ and thus $\gamma(e)(\nu)$ is satisfied. We define ν'' by: $\nu'' = \nu_{\lceil \text{CUT}(K) \rceil}$ and $\nu''(\delta_e) = \nu(\text{CUT}(K))$ and $\nu''(\delta_p) = \nu(\delta_e)$ for $p \in e^\bullet$. Because $\nu \in Z \cap \llbracket \text{GUARD}(\lambda(e)) \rrbracket$ it implies $\nu[\text{RESET}(\lambda(e))]$ is in $\llbracket \text{INV}(\mathbf{I}) \rrbracket$. This in turn implies $\nu'' \in \llbracket \Phi_{\text{CUT}(K')} \rrbracket$. Now define ν' by: $\nu'_{\lceil K' \rceil} = \nu''$ and $\nu'(\delta_p) = \nu''(\delta_e) + \delta$ for $p \in \text{CUT}(K')$. Using the downward closure of invariants and the fact that $v' \in \llbracket \text{INV}(\mathbf{I}) \rrbracket$ we obtain that $\nu' \in \llbracket \Phi_{\text{CUT}(K')} \rrbracket$. Hence $\text{GS}(K') = (\mathbf{1}', Z')$.

Now it remains to prove that $\text{GS}(\{\perp\}) = (\mathbf{1}_0, Z_0)$. Let $K = \{\perp\}$. Assume $(M, v) \in \text{GS}(K)$; It is easy to see that $M = (\ell_{1,0}, \dots, \ell_{n,0})$. By definition of $\text{GS}(K)$, $v(x) = \nu(\text{CUT}(K)) - \nu(\delta_{Z_{\{\perp\}}^{\neq 0}(x)}) = \nu(\text{CUT}(K))$ because $\nu \in \Phi_{\text{CUT}(K)}$, with $\nu(\text{CUT}(K)) = \nu(\delta_{(\perp, l_{i,0})})$ for each $1 \leq i \leq n$. This implies $v(x_1) = v(x_2) = \dots = v(x_n)$. As $\nu \in \Phi_{\text{CUT}(K)}$, the Φ_1 part implies $\text{INV}(l_{i,0})[x \mapsto \delta_{(\perp, l_{i,0})}](\nu)$ holds and this is also the value of $\text{INV}(l_{i,0})(v_{X_i})$; hence $v \in Z_0$. Now assume $v \in Z_0$. Define $\nu(\delta_{(\perp, l_{i,0})}) = v(x)$ for any $x \in X$ and $\nu(\perp) = 0$. By the same argument as before on invariants, $\text{INV}(l_{i,0})[x \mapsto \delta_{(\perp, l_{i,0})}](\nu)$ holds and thus $\nu \in \llbracket \Phi_{\text{CUT}(K)} \rrbracket$.

This completes the proof of Theorem 1. \square

As a consequence of Theorem 1, we obtain that each Z_K is a zone for a configuration K . Actually, this zone can be computed as follows: let K be a cut and fix δ_p with $p \in \text{CUT}(K)$.

Then $Z_K = \llbracket \Phi_{\text{CUT}(K)}[\delta_{Z_K^{\neq 0}(x_k)} \mapsto \delta_p - x_k] \rrbracket_X$. The substitution we use is a simple substitution and thus we obtain a formula in $\mathcal{C}(X \cup \delta(\lceil \text{CUT}(K) \rceil))$ the projection of which is in $\mathcal{C}(X)$. This gives an effective procedure to compute Z_K and we will use it in the next subsection. Note also that it is possible to check inclusion between zones as well as equality. This enables us to check equality of symbolic global states.

We use the term *weak correctness* in the theorem, because it relies on a predicate $\Phi_{\text{CUT}(K)}$ which contains constraints not necessarily obtained with $\lceil K \rceil$. Section 4 will deal with this issue.

We denote $\eta(e) = \gamma(e) \wedge (\bigwedge_{p \in \bullet_e} \delta_p = \delta_e)$. A corollary of Theorem 1 is:

Corollary 1. *Let $K = \{e_1, \dots, e_j\}$ be a configuration s.t. $\nu \in \llbracket \Phi_{\text{CUT}(K)} \wedge (\bigwedge_{1 \leq i \leq j} \eta(e_i)) \rrbracket \neq \emptyset$. Then there exists a one-to-one mapping $f : [1..j] \rightarrow [1..j]$ s.t. $(\lambda(e_{f(1)}), \nu(e_{f(1)})) \dots (\lambda(e_{f(j)}), \nu(e_{f(j)}))$ is a timed word accepted by the NTA $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$.*

Proof. The proof is by induction on $|K|$ and is a direct application of Theorem 1 and the properties of the simulation graph. \square

Theorem 2 (Completeness). *Let $(\mathbf{1}, Z)$ be a reachable symbolic state in the simulation graph of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$. There is a configuration K of the underlying net of $\text{TBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ s.t.:*

- $\text{GS}(K) = (\mathbf{1}, Z)$ and,
- if $(\mathbf{1}, Z) \xrightarrow{t} (\mathbf{1}', Z')$ there is a configuration $K \cup \{e\}$, with $\lambda(e) = t$ and $\text{GS}(K \cup \{e\}) = (\mathbf{1}', Z')$.

Proof. We prove the following statement (S): assume $(\mathbf{1}, Z)$ is reachable in the simulation graph, and $(\mathbf{1}, Z) \xrightarrow{t} (\mathbf{1}', Z')$. Then if K is a configuration and $\text{GS}(K) = (\mathbf{1}, Z)$, $K \cup \{e\}$ is a configuration and $\text{GS}(K \cup \{e\}) = (\mathbf{1}', Z')$. If we are able to prove that $\text{GS}(\{\perp\}) = (\mathbf{1}_0, Z_0)$ the results of Theorem 2 follow. This last statement is true and the proof is simple and similar to the initial case of the proof of Theorem 1.

To prove (S) we proceed as follows: 1) prove that $K \cup \{e\}$ is a cut in $\text{TBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ i.e. $\llbracket \Phi_{\text{CUT}(K \cup \{e\})} \rrbracket \neq \emptyset$ and 2) prove that $\text{GS}(K \cup \{e\}) = (I', Z')$.

As $\text{GS}(K) = (I, Z)$ and (I, Z) is in the simulation graph, $Z \neq \emptyset$. Moreover there exists some $v \in \llbracket \text{GUARD}(t) \rrbracket \cap Z$. For such a v there exists $\nu \in \llbracket \Phi_{\text{CUT}(K)} \rrbracket$ s.t. $v(x) = \nu(\text{CUT}(K)) - \nu(\delta_{Z_K^0}(x))$. First notice that there is a finite symbolic process (E, P) for $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ that contains a configuration $K \cup \{e\}$ but it might be the case that $\llbracket \Phi_{\text{CUT}(K \cup \{e\})} \rrbracket = \emptyset$. To prove that $K \cup \{e\}$ is a configuration of $\text{TBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ we just need to show that $\llbracket \Phi_{\text{CUT}(K \cup \{e\})} \rrbracket \neq \emptyset$ as, in this case, it is not removed from the maximal branching process and thus is a cut of $\text{TBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$. As $K \cup \{e\}$ is a configuration of (E, P) we can extend the valuation ν to $\{e\} \cup e^\bullet$ and set: $\nu(\delta_e) = \nu(\text{CUT}(K))$ and $\nu(\delta_p) = \nu(\text{CUT}(K))$ for $p \in e^\bullet$. Now we can prove that $\gamma(e)(\nu)$ holds. Indeed, $\gamma(e)(\nu) = \text{GUARD}(t)[x \mapsto \delta_e - \delta_{Z_{K \cup \{e\}}^0}(x)](\nu)$ and this is equal to $\text{GUARD}(t)[x / (\nu(\delta_e) - \nu(\delta_{Z_K^0}(x)))]$. As $\nu(\delta_e) = \nu(\text{CUT}(K))$, $\nu(\delta_e) - \nu(\delta_{Z_K^0}(x)) = v(x)$ and thus $\gamma(e)(\nu)$ holds.

To prove that $\nu \in \llbracket \Phi_{\text{CUT}(K \cup \{e\})} \rrbracket$ we just have to prove that $\gamma(p)(\nu)$ holds for each $p \in e^\bullet$ as the other statements are true because $\nu \in \Phi_{\text{CUT}(K)}$. In the definition of timed automata we use, we have the following: $\text{GUARD}(t)(v)$ holds implies $\text{INV}(1)(v[\text{RESET}(t)])$ holds. This means that for each automaton involved in the transition, i.e. $p \in \text{Loc}(e^\bullet)$, we have $\text{INV}(\text{loc}(p))(v[\text{RESET}])$ holds. Now it suffices to notice that if $x \in \text{RESET}(t)$ then $x \in \text{RESET}(e)$ and $\nu(\text{CUT}(K \cup \{e\})) = \nu(\delta_{Z_{K \cup \{e\}}^0}(x)) = v[\text{RESET}(t)](x)$. Otherwise $\nu(\text{CUT}(K \cup \{e\})) = \nu(\text{CUT}(K \cup \{e\}))$ and $\nu(\delta_{Z_{K \cup \{e\}}^0}(x)) = \nu(\delta_{Z_K^0}(x))$ and thus $\nu(\text{CUT}(K \cup \{e\})) - \nu(\delta_{Z_{K \cup \{e\}}^0}(x)) = \nu(\text{CUT}(K \cup \{e\})) - \nu(\delta_{Z_K^0}(x)) = v(x)$. As $\gamma(p)(\nu) = \text{INV}(\text{loc}(p))[x / (\nu(\text{CUT}(K \cup \{e\})) - \nu(\delta_{Z_{K \cup \{e\}}^0}(x)))]$, this is equal to $\text{INV}(\text{loc}(p))[x / v[\text{RESET}(t)](x)]$ and thus $\gamma(p)(\nu)$ holds for each $p \in e^\bullet$. Hence $\nu \in \llbracket \Phi_{\text{CUT}(K \cup \{e\})} \rrbracket$ and this proves 1).

To prove 2) we have to show: i) if $v' \in \llbracket \Phi_{\text{CUT}(K \cup \{e\})} \rrbracket$ then $v' \in Z'$ and ii) if $v' \in Z'$ then $v' \in \llbracket \Phi_{\text{CUT}(K \cup \{e\})} \rrbracket$. This consists in building valuations exactly as in the last part of the proof of Theorem 1 and we re-use the construction of this part of the proof.

This completes the proof. \square

A corollary of Theorem 2 is the following:

Corollary 2. *Let $(w_1, d_1)(w_2, d_2) \dots (w_k, d_k)$ be a timed word accepted by the NTA $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$. Then, there exists a configuration $K = \{e_1, \dots, e_k\}$ of $\text{TBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ and $\nu : \llbracket \text{CUT}(K) \rrbracket \rightarrow \mathbb{R}_{\geq 0}$, $\nu \in \llbracket \Phi_{\text{CUT}(K)} \rrbracket$, s.t. 1) $\lambda(e_i) = w_i$ and 2) $\nu(\delta_{e_i}) = d_i$.*

Proof. The basic step of the proof has already been done in the proof of Theorem 2 when we construct a new valuation on $K \cup \{e\}$. The proof of the corollary uses the same construction and an induction step. \square

3.3 Complete Finite Prefix for Network of Timed Automata

If a TBP \mathcal{T} satisfies the conditions of Theorem 2, we say that \mathcal{T} is *complete*. Theorem 4, corresponds to a *correctness* property. For network of finite automata,

complete (and correct) finite branching processes exist, and are called *complete finite prefixes* [24,14]. In the case of network of timed automata we can construct of finite complete prefix that preserves the reachability information of the simulation graph.

The reason for the existence of such a finite prefix is that each symbolic cut corresponds to a symbolic state in the simulation graph of the network of timed automata and the number of symbolic states is finite (the timed automata are bounded).

This implies that to construct a symbolic complete finite prefix that contains at least as much information as the simulation graph, we can re-use the algorithms of [14] and the notion of *cut-off* events and *adequate orders*. Adequate orders are defined in [14] and we refer the reader to this article for a comprehensive definition and list of the properties of these orders. The notion of *cut-off* events is given w.r.t. adequate orders:

Definition 10 (Cut-off events [14]). *Assume \prec is an adequate order on the configurations. Let (E, P, γ) be a symbolic branching process s.t. $e \in E$. e is a cut-off event w.r.t. \prec if there exists an event $e' \in E$ s.t. $[e'] \prec [e]$ and $\text{GS}([e]) = \text{GS}([e'])$.*

The idea behind a *cut-off* event is that the part of the (symbolic) branching process that appears under (or after) the cut-off event need not be developed because it is isomorphic to a part of the branching process already obtained⁹ and thus will give no more information. Adequate orders can be obtained from orders on the set $(T_1^\varepsilon)^* \times \dots \times (T_n^\varepsilon)^*$ if the sets T_i are the sets of transitions of each automaton. As we have an effective procedure to check equality of symbolic global states, we can use the algorithm of [14] with the different adequate orders they propose.

As a consequence symbolic complete prefixes exist for network of timed automata and can be computed using the efficient algorithms of [14]. We let $\text{PREFIX}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$ be the complete symbolic finite prefix obtained from $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$.

A complete finite prefix for the example of Fig. 1.(b) is given on Fig. 2. Note that it is trivially finite because the symbolic occurrence net associated with it is finite so we do not need to use any adequate order for this example. As an example, the global symbolic state for the configuration $\{\perp, e_2\}$ is $((0, B, V), Z_{\{\perp, e_2\}})$ with $Z_{\{\perp, e_2\}} = \llbracket \varphi \rrbracket_X$ and

$$\begin{aligned} \varphi = & (\delta_\perp = 0) \wedge (\delta_U \leq 3) \wedge (\delta_B - \delta_{e_2} \leq 2) \\ & \wedge (\delta_0 \geq \delta_\perp) \wedge (\delta_A \geq \delta_\perp) \wedge (\delta_U \geq \delta_\perp) \\ & \wedge (\delta_A = \delta_{e_2}) \wedge (\delta_U = \delta_{e_2}) \wedge (\delta_0 = \delta_B = \delta_V) \\ & x = \delta_0 - \delta_{e_2} \wedge y = \delta_0 - \delta_\perp \end{aligned} \tag{5}$$

which gives $Z_{\{\perp, e_2\}} = \llbracket y - x \leq 3 \wedge x \leq 2 \rrbracket$.

⁹ Actually it is only when the construction of the prefix is completed that this property holds.

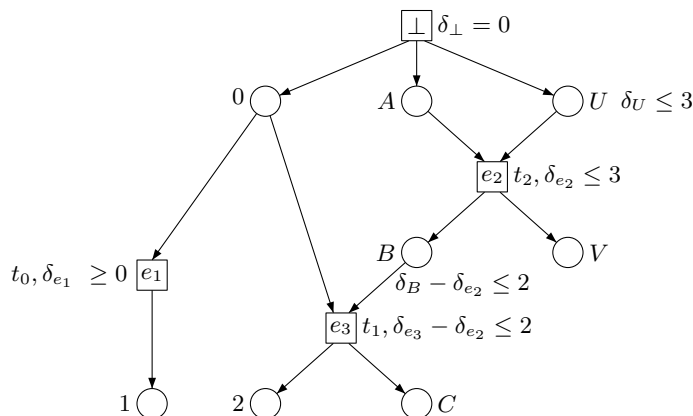


Figure 3. Symbolic unfolding for the example of Fig. 2

The example of Fig. 4, page 18 has an infinite unfolding (synchronization is on common labels). A complete prefix is given on Fig. 4.(a), page 18. We use the order \prec_1 of [14] to generate this complete finite prefix, and event e_4 is a cut-off event.

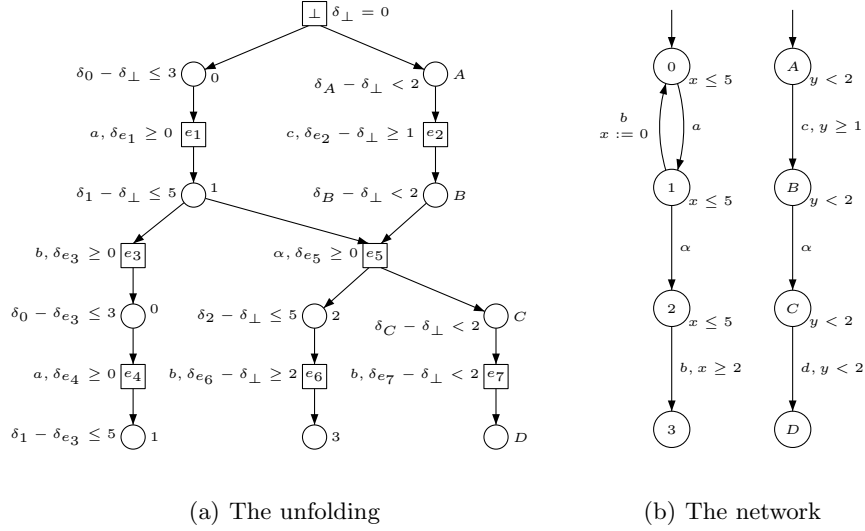
Note that our SBP does not solve Problem 2, as Problem 2 requires to have: if $K = \{e_1, \dots, e_k\}$ is a configuration of (E, P) and $\nu : [K] \rightarrow \mathbb{R}_{\geq 0}$ s.t. $\gamma(\nu)$ holds on $[K]$ then $\{(e_{f(1)}, \nu(\delta_{e_{f(1)}})), \dots, (e_{f(k)}, \nu(\delta_{e_{f(k)}}))\}$ is a timed word with $f : [1..k] \rightarrow [1..k]$ a one-to-one mapping.

For instance, in the unfolding of Fig. 3, e_1 can occur before e_2 if e_1 occurs before time 3. But there is another property relating the dates at which e_1 and e_2 occur: if e_2 occurs before e_1 , e_1 can only occur within 2 time units after e_2 . This is not captured by the constraints on the e_1 and e_2 in our SBP.

In the next section we refine the SBP to obtain an *extended* SBP that satisfies the previous requirement.

4 Extended Finite Complete and Correct Prefixes

In the case of finite automata, any cut containing a co-set that enables an event, still enables the same event. This is not the case for network of timed automata as can be seen on the example of Fig. 2. Assume we know that e_2 has not fired. Then e_1 can fire because nothing can prevent it from doing so (e_3 is not enabled). The fact that e_2 has not fired can be inferred from the fact that either place A or U contains a token. But this implies a constraint on the firing time of e_1 , *i.e.* $\delta_{e_1} \leq 3$, and this constraint can be inferred from $[e_1]$. If e_2 has fired at δ_{e_2} , e_3 and e_1 are in conflict. Thus e_1 can only occur at a date when a token can be in B , *i.e.* $\delta_{e_1} - \delta_{e_2} \leq 2$. Thus timing constraints among events are not the same in the cuts $(0, A, U)$ and $(0, B, V)$, and they are both cuts that contain $\bullet e_1$.



(a) The unfolding

(b) The network

Figure 4. A network with a loop

To encode this timing dependency structurally we can use symbolic occurrence nets with *read arcs*. For instance the symbolic net of Fig. 2 can be “transformed” into the symbolic extended net of Fig. 5 (a read arc is a dash line). In this extended occurrence net, the constraint between the dates of occurrences of e_1 and e_2 can be inferred from the sole past of e_1 (that includes the read places and the formula $\Phi_{\bullet e_1 \cup \circ e_1}$): indeed, to fire, we must have $\delta_{e_1} = \delta_B$ (because there is a read arc from B to e_1) and thus $\delta_{e_1} - \delta_{e_2} \leq 2$. Read arcs enable us to differentiate the two cuts $(0, A, U)$ and $(0, B, V)$ that generate different timing constraints on the firing time of e_1 .

We first define *extended nets* that are nets with read arcs, and then show how to build an extended net that is a complete finite prefix for a NTA.

4.1 Extended Nets

Definition 11 (Extended Nets). An extended net \mathcal{N} is a tuple $(E, P, \bullet(), ()^\bullet, ()^\circ)$ where $(E, P, \bullet(), ()^\bullet)$ is a net, and $()^\circ : E \rightarrow 2^P$. If $\circ e = \emptyset$ for each $e \in E$ then \mathcal{N} is a net.

The set $\circ e$ represents the input places of an event that are to be read without removing a token.

The causality relation is now defined by: $x \rightarrow y$ if $x \in \bullet y \cup \circ y$ or $y \in x^\bullet$. \preceq is the reflexive and transitive closure of \rightarrow . The *weak causality relation* \dashrightarrow

is given by: $x \dashrightarrow y$ if either $x \rightarrow y$ or ${}^\circ x \cap \bullet y \neq \emptyset$ (if x needs a token in one of the input place of y this implies a causality relation, even if x is not in the past of y in the sense of \rightarrow). We let \preceq the reflexive and transitive closure of \dashrightarrow . Two nodes x and y are *weakly causally related* if either $x \preceq y$ or $y \preceq x$. x and y are in conflict, $x \# y$, if there is a place p s.t. there exist w and u , $w \neq u$, $p \in \bullet u \cap \bullet w$ and $w \preceq x$ and $u \preceq y$. x and y are concurrent if they are not weakly causally related nor in conflict. For $J \subseteq E \cup P$, the definitions of $\uparrow J$ and $\lceil J \rceil$ are unchanged (we use the new \preceq). A set of events is now causally closed if $\lceil J \rceil = J$. Co-sets are now defined with the extended concurrency relation. Configurations and cuts are defined as before.

An extended symbolic net is a tuple $(E, P, \bullet(), ()^\bullet, {}^\circ(), \gamma)$ with $(E, P, \bullet(), ()^\bullet, {}^\circ())$ an extended nets and γ has the properties of Definition 7 (using the new past operator $\lceil \cdot \rceil$). The semantics of extended occurrence nets requires that $\bullet e \cup {}^\circ e \subseteq M$ to fire event e from marking M .

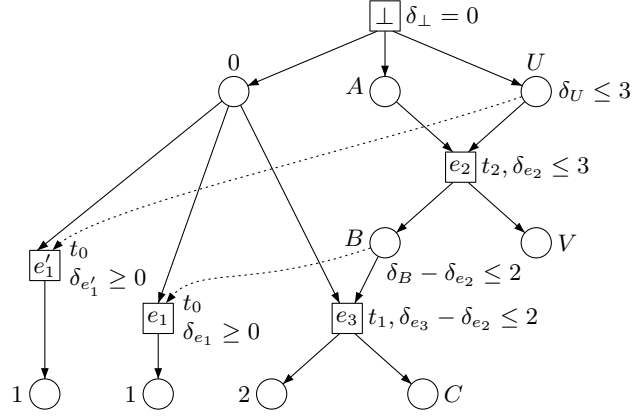


Figure 5. Extended symbolic unfolding for the example of Fig. 1

The *Extended* symbolic branching processes (ESBP) of a network are defined as in section 3: the only change we need to do is to define the set of events so that it includes the places in ${}^\circ e$. To this end, if $S, S' \subseteq \mathcal{P}$ and $t \in T$, (S, S', t) is in \mathcal{E} and if $e = (S, S', t)$, ${}^\circ e = S'$.

4.2 Safe Co-sets

We now define *safe co-sets*: there are extended co-sets that contain the minimal information required about places in the net to ensure an event can fire.

Time Lock Freedom. The *time lock freedom* assumption TL , asserts that no automaton in the NTA can prevent time from elapsing. Under this assumption,

if an event e is not in conflict with any other event e' , the timing constraint on δ_e obtained by $\Phi_{\bullet e}$ is sufficient to ensure that e can fire at time δ_e . Indeed, the only way an event e can be prevented from happening at time t is because either i) an event in conflict with it occurred at time $t' < t$, or ii) the NTA cannot reach time t . Under the assumption of time lock freedom, ii) cannot happen and consequently if e is not in conflict with any e' , the constraints on $\lceil \bullet e \rceil$ are sufficient to ensure e can fire.

If we don't have this assumption, the level of concurrency in the unfolding is reduced as witnessed by the examples of appendix A.

Notice that we can weaken the time lock freedom assumption TL as follows obtaining TL' : for any event e , if e is not in conflict with any e' , then the automata which are not involved in $\bullet e$ cannot prevent time from elapsing.

The NTA of Fig. 1 does not satisfy the time lock freedom assumption TL (automaton 2 can prevent time from elapsing in location B) but satisfy TL' (the first automaton cannot prevent time from elapsing).

Safe Co-sets. Let $\text{ENABLE}(e)$ denote the *enabling cuts* of $e \neq \perp$ in a finite symbolic branching process \mathcal{N} :

$$\text{ENABLE}(e) = \{C \mid \bullet e \subseteq C \text{ and } C \text{ is a cut of } \mathcal{N}\}.$$

As a running example we take the prefix \mathcal{N}_1 built in Fig. 2 and δ_\perp is always replaced by 0 (zero). For this example the enabling sets are:

$$\begin{aligned} \text{ENABLE}(e_1) &= \{(0, A, U), (0, B, V)\} \\ \text{ENABLE}(e_2) &= \{(0, A, U), (1, A, U)\} \\ \text{ENABLE}(e_3) &= \{(0, B, V)\}. \end{aligned}$$

Now assume an event e is in conflict with another event e' in the symbolic unfolding. As we pointed out at the end of section 3, the timing constraints given by $\lceil \bullet e \rceil$ on the firing time of e do not always contain enough information to ensure event e can fire: event e_1 in \mathcal{N}_1 can fire if a) e_2 has not fired (at time $\delta \leq 3$), or b) e_2 has fired, and the time elapsed is less than 2 time units (*i.e.* at time δ with $\delta - \delta_{e_2} \leq 2$), or c) e_2 has been disabled by another event in conflict with it and cannot occur in the future. To ensure e can fire, we should add to the conditions in $\bullet e$ some information about the events in conflict with e . This is the purpose of *safe co-sets*. They extend the co-sets of the symbolic unfolding with some information about the conflicting events. In terms of Petri nets, a safe co-set for an event e will be the set of places $\bullet e$, extended with a set a *read only* places, ${}^\circ e$. The information contained in a safe co-set should be such that, if the timing constraints obtained by $\Phi_{\bullet e \cup {}^\circ e}$ are satisfied by δ_e , then e can fire at time δ_e .

For any cut C , the formula Φ_C (Definition 8, equations (1)–(4)) is a formula over $\delta(C \cup (\lceil C \rceil \cap E))$. Indeed all the intermediate places p , not in the cut, are constrained by a formula of the form $\delta_e = \delta_p$ because of equation 2 of Definition 8. For instance $\Phi_{(0, B, V)} = \delta_B - \delta_{e_2} \leq 2 \wedge 0 \leq \delta_{e_2} \leq 3 \wedge \delta_B \geq \delta_{e_2} \wedge \delta_0 = \delta_B = \delta_V$.

Also because of the term Φ_4 , if we use an extra variable δ and the formula $(\delta = \delta_p) \wedge \Phi_C$ for any¹⁰ $p \in C$, we obtain a formula over $\delta(\lceil C \rceil \cap E) \cup \{\delta\}$: δ stands for the current time (since the system started) and the constraint on δ in Φ_C defines the set of instants for which the cut C is reachable. We write Φ_C^δ for the projection on $(\lceil C \rceil \cap E) \cup \{\delta\}$ of the formula $\Phi_C \wedge (\delta = \delta_p)$. In our example, $\Phi_{(0,A,U)}^\delta = \delta \leq 3$, $\Phi_{(1,A,U)}^\delta = \delta_{e_1} \leq 3 \wedge \delta_{e_1} \leq \delta \leq 3$ and $\Phi_{(0,B,V)}^\delta = \delta - \delta_{e_2} \leq 2 \wedge 0 \leq \delta_{e_2} \leq 3 \wedge \delta \geq \delta_{e_2}$. We let $\Theta(e) = \{\Phi_C^\delta \mid C \in \text{ENABLE}(e)\}$. In our example, we obtain:

$$\begin{aligned}\Theta(e_1) &= \{\Phi_{(0,A,U)}^\delta, \Phi_{(0,B,V)}^\delta\} \\ \Theta(e_2) &= \{\Phi_{(0,A,U)}^\delta, \Phi_{(1,A,U)}^\delta\} \\ \Theta(e_3) &= \{\Phi_{(0,B,V)}^\delta\}.\end{aligned}$$

$\Theta(e)$ represents the set of different constraints that can be generated by all the enabling cuts of event e . We also need to define the set of places from which we can be sure that an event e will not fire (because an event in conflict with e has occurred). We let $\text{DEAD}(e)$ be the set defined by: $p \in \text{DEAD}(e)$ iff for each configuration K , $p \in \lceil \text{CUT}(K) \rceil$ implies $e \notin K$. In words, if a configuration is such that place p has received a token, then e cannot occur any more as no configuration containing e and p exists.

Definition 12 (Safe Representatives). *A set of places S is a safe representative of e w.r.t. e' if*

$$\bullet e \subseteq S \tag{6}$$

$$S \cap \lceil e' \rceil \neq \emptyset \text{ or } S \cap \text{DEAD}(e') \neq \emptyset \tag{7}$$

$$\forall C \in \text{ENABLE}(e), S \subseteq C \implies \llbracket \Phi_S^\delta \rrbracket = \llbracket \Phi_C^\delta \rrbracket. \tag{8}$$

S is a safe representative of e if S is a safe representative of e w.r.t. e' for any e' s.t. $e \# e'$.

If S is a safe representative of e , then S contains enough information to infer the constraints on the firing time of e for any cut C s.t. $S \subseteq C$. For instance $(0, B)$ is a safe representative of e_1 (w.r.t. to e_3 , but e_3 is the only event in conflict with e_1). $(0, U)$ is a safe representative of e_1 as well as $(0, A, U)$. $(0, B)$ is a safe representative of e_3 .

Definition 13 (Complete Set of Safe Representatives). *A set \mathcal{S} is a complete set of safe representatives for e if:*

1. each $S \subseteq \mathcal{S}$ is a safe representative of e ;
2. for each cut $C \in \text{ENABLE}(e)$, there is some $S \in \mathcal{S}$ s.t. $S \subseteq C$.

¹⁰ As equation (4) already imposes $\delta_{p'} = \delta_p$ for $p, p' \in C$ we can add $\delta = \delta_p$ for any p in C .

As each cut $C \in \text{ENABLE}(e)$ is a safe representative of itself, there is at least one complete set of safe representatives which is $\text{ENABLE}(e)$.

We can state a theorem which is variant of Theorem 1 using only safe representatives of an event (item 2 of the theorem):

Theorem 3 (Correctness). *If K is a configuration of $\text{TBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ s.t. $\text{GS}(K) \neq \emptyset$ then*

1. $\text{GS}(K) = (\mathbf{1}, Z)$ for some $(\mathbf{1}, Z)$ reachable in $\text{SG}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$,
2. if $K \cup \{e\}$ is a configuration and S is a safe representative of e s.t. $\llbracket \Phi_S \wedge \gamma(e) \wedge (\bigwedge_{p \in \bullet_e} \delta_p = \delta_e) \rrbracket \neq \emptyset$ then $(\mathbf{1}, Z) \xrightarrow{\lambda(e)} (\mathbf{1}', Z')$ and $\text{GS}(K \cup \{e\}) = (\mathbf{1}', Z')$.

Proof. The proof follows directly from Theorem 1 and the definition of safe representatives. \square

This theorem states that a safe representative for e contains enough information to decide whether event e can be fired or not. As a consequence, if whenever we attach a new event e to a finite branching process of a NTA $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$, we add *read-arcs* to the places of a safe representative of e , then $\lceil e \rceil$ gives the accurate constraints on the date δ_e at which e can fire.

Minimality for Safe Co-sets. The purpose of unfoldings is to keep explicit the maximal amount of concurrency. In the case of untimed network of automata, \bullet_e is sufficient to ensure e can fire. For NTA, we have to use read arcs, but we should be concerned about the range of the new dependencies: for instance, if we use $\text{ENABLE}(e)$ as complete set of safe representatives for each e , we require that the global state of the network is known each time we want to fire e . This means we do not keep explicit any concurrency in the unfolding. It is thus important to try and reduce the number of read arcs from each event. To this extent we define a notion of *minimality* for complete sets of safe representatives.

First, a complete set of safe representatives $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$ for e is *non redundant* if each cut $C \in \text{ENABLE}(e)$ is represented at most once: $\forall C \in \text{ENABLE}(e), C \subseteq S_i \text{ and } C \subseteq S_j \implies i = j$. $\text{ENABLE}(e)$ is a non redundant complete set of safe representatives. Each non redundant set \mathcal{S} induces a partitioning of $\text{ENABLE}(e)$ denoted $\text{ENABLE}(e)_{/\mathcal{S}}$. The class of an element $S \in \mathcal{S}$ is denoted $[S]$.

Let \mathcal{S}_1 and \mathcal{S}_2 be two non redundant sets. We define the (partial) order \sqsubseteq by: $\mathcal{S}_1 \sqsubseteq \mathcal{S}_2$ if $\text{ENABLE}(e)_{/\mathcal{S}_1} = \text{ENABLE}(e)_{/\mathcal{S}_2}$ and for each $S \in \mathcal{S}_1, S' \in \mathcal{S}_2$ s.t. $[S] = [S']$ we have $|S| \leq |S'|$.

Given $e \in E$, $\text{SAFE}(e)$ denotes a minimal complete set of safe representatives for all the $C \in \text{ENABLE}(e)$. In the example for \mathcal{N}_1 we can take the sets:

$$\begin{aligned} \text{SAFE}(e_1) &= \{(0, U), (0, B)\} \\ \text{SAFE}(e_2) &= \{(A, U)\} \\ \text{SAFE}(e_3) &= \{(0, B)\} \end{aligned}$$

Notice that if an event e is not in conflict with any other event (like e_2 in \mathcal{N}_1), $\bullet e$ is a minimal complete set of safe representatives¹¹. Also, the minimality criterion we have defined does not give a unique set of complete safe representatives. A consequence in section 4.3 is that extended complete finite prefixes for NTA are not defined uniquely, and there might be many different extended prefixes obtained from the same NTA.

4.3 Extended Finite Branching Process

We can now build a parameterized version of extended finite branching process: assume O is an *oracle* defined on $T \times \mathcal{P}$ s.t. $O(C, t) = \{C_1, C_2, \dots, C_l\}$ where the C_i 's are subset of \mathcal{P} .

Definition 14 (Extended Finite Branching Processes). *The O -extended finite branching processes (EFBP) of a network $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ are defined inductively as follows:*

- $(\{\perp\}, \{(\perp, l_{1,0}), (\perp, l_{2,0}), \dots, (\perp, l_{n,0})\}, \gamma)$ with $\gamma(\perp) \stackrel{def}{=} \delta_\perp = 0$, $\gamma(\perp, l_{i,0}) \stackrel{def}{=} \text{INV}(l_{i,0})[s_i]$, with the simple substitution s_i defined by $s_i(x) = \delta_{(\perp, l_{i,0})}$ for each $x \in X_i$, is an O -EFBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$;
- if (E, P, γ) is an O -EFBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$, $t \in T$, $C \cup C'$ is a co-set s.t. $\text{Loc}(C) = \text{SRC}^*(t)$, $C' \in O(C, t)$ then

$$(E \cup \{e\}, P \cup \{(e, s) \mid s \in \text{TGT}^*(t)\}, \gamma')$$

with $e = (C, C', t)$, $\gamma'_{|E \cup P} = \gamma$, $\gamma'(e) = \text{GUARD}(t)[\eta]$, $\eta : x \mapsto \delta_e - \delta_{Z_{\overline{C}}=0}(x)$, $\gamma'(e, s) = \text{Inv}(s)[\eta']$, $\eta' : x \mapsto \delta_{(e,s)} - \delta_{Z_{\overline{C}}=0}(x)$, is a SBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$.

If $e \notin E$, e is a possible extension of (E, P) .

ESBP are again closed under countable union and we define the Extended unfolding of a network to be the maximal element of this set and denote it $\text{EBP}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$.

If (E, P) is an EFBP, we can define a SBP from (E, P) by simply removing the read-arcs. To do this, we define the mapping $\omega : E \cup P \rightarrow E \cup P$ by: $\omega(\perp) = \perp$, $\omega(S, S', t) = (\omega(S), t)$ and $\omega(e, s) = (\omega(e), s)$ and denote $\omega(E, P)$ the SBP obtained this way. For any EFBP (E, P) , $\omega(E, P)$ is an SBP of $\text{TBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$.

As for SBP, the mapping γ is defined in an unambiguous manner for EFBP given the set of places and events. Also note that the constraint $\gamma(\omega(x))$ is exactly $\gamma(x)[s]$ where $s : \delta_x \mapsto \delta_{\omega(x)}$. We denote $\omega(\gamma)$ the mapping that associates to each constraint $\gamma(x)$ the constraint $\gamma(\omega(x))$.

Hence the following proposition holds:

Proposition 3. *If (E, P, γ) is an EFBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$, $(\omega(E), \omega(P), \omega(\gamma))$ is a SBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$.*

¹¹ Under the assumption of time lock freedom

This entails that $\omega(\text{EBP}(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$ is equal to $\text{TBP}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$.

To define the oracle O , we use $\omega(E, P)$: let (C, t) be an event in an EFBP of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$:

$$O(C, t) = \{C' \subseteq \mathcal{P} \mid \omega(C \cup C') \in \text{SAFE}((\omega(C), t))\}$$

In other words, this is equivalent to saying: to add event $e = (C, t)$ in the EFBP, a safe co-set of e should be used. As the definition of safe co-sets is not unique (because the definition of minimal complete sets of safe representatives is not) the EFBP depends on the sets of safe representatives of each event.

4.4 Extended Finite Complete Prefix

We now define the symbolic *configurations* of an extended net \mathcal{T} :

Definition 15 (Symbolic Configuration). (K, Ψ) is a symbolic configuration of \mathcal{T} if:

1. K is a cut of the underlying net,
2. $\Psi = \Psi_1(K) \wedge \Psi_2(K)$ where $\Phi_i(M), 1 \leq i \leq 2$ are defined by:

$$\Psi_1(K) = \bigwedge_{e \in \lceil K \rceil} \gamma(e) \tag{9}$$

$$\Psi_2(K) = \bigwedge_{e \in K} (\bigwedge_{p \in \bullet e} \delta_p = \delta_e) \tag{10}$$

Notice that $\Psi(K)$ uses only information in the past of K . We can re-write Theorem 3 as follows:

Theorem 4 (Correctness). If K is a configuration of $\text{EBP}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$ s.t. $\llbracket \Psi(K) \rrbracket \neq \emptyset$ then:

1. $\text{GS}(K) = (\mathbf{1}, Z)$ for some $(\mathbf{1}, Z)$ reachable in $\text{SG}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$,
2. if $K \cup \{e\}$ is a configuration and $\llbracket \Psi(K \cup \{e\}) \rrbracket \neq \emptyset$ then $(\mathbf{1}, Z) \xrightarrow{\lambda(e)} (\mathbf{1}', Z')$ and $\text{GS}(K \cup \{e\}) = (\mathbf{1}', Z')$.

Proof. We prove the following: if 1) $\llbracket \Psi(K) \rrbracket \neq \emptyset$ implies $\llbracket \Phi_{\omega(K)} \rrbracket \neq \emptyset$ and 2) $\text{GS}(K) = \text{GS}(\omega(K)) = (\mathbf{1}, Z)$ is reachable in the simulation graph, then i) $\llbracket \Psi(K \cup \{e\}) \rrbracket \neq \emptyset$ implies $(\mathbf{1}, Z) \xrightarrow{\lambda(e)} (\mathbf{1}', Z')$ and ii) $\text{GS}(K \cup \{e\}) = (\mathbf{1}', Z')$. This together with $\llbracket \Psi(\{\perp\}) \rrbracket \neq \emptyset$ implies $\llbracket \Phi_{\omega(\{\perp\})} \rrbracket \neq \emptyset$ and $\text{GS}(\{\perp\}) = (\mathbf{1}_0, Z_0)$ implies Theorem 4.

Assume $\llbracket \Psi(K) \rrbracket \neq \emptyset$ implies $\llbracket \Phi_{\omega(K)} \rrbracket \neq \emptyset$ and $\text{GS}(K) = (\mathbf{1}, Z)$. It suffices to remark that $\text{GS}(K) = \text{GS}(\omega(K))$ and using point 1 of Theorem 3 we obtain $\text{GS}(K) = (\mathbf{1}, Z)$ for some $(\mathbf{1}, Z)$ reachable in the simulation graph.

Now assume $\llbracket \Psi(K \cup \{e\}) \rrbracket \neq \emptyset$. In the SBP, $\omega(\bullet e \cup \circ e)$ is a safe representative of $\omega(e)$. Also in the EBP, $\bullet e \cup \circ e \subseteq \text{CUT}(K)$ otherwise $K \cup \{e\}$ is not a configuration

(remind we use the definition of configurations for extended nets). Hence $\omega(\bullet e \cup \circ e) \subseteq \text{CUT}(\omega(K))$ and thus $\omega(\bullet e \cup \circ e)$ is a safe representative of $\text{CUT}(\omega(K))$.

It follows that $\Phi_{\text{CUT}(\omega(K))}$ is equivalent to $\Phi_{\text{CUT}(\omega(\bullet e \cup \circ e))}$ (property of safe representatives). $\Psi(K \cup \{e\})$ is exactly $\varphi = \Phi(\omega(\bullet e \cup \circ e)) \wedge \gamma(\omega(e)) \wedge (\bigwedge_{p \in \bullet \omega(e)} \delta_p = \delta_{\omega(e)})$ up to renaming of the δ_x in $\delta_{\omega(x)}$, and $\llbracket \Psi(K \cup \{e\}) \rrbracket \neq \emptyset$ implies $\llbracket \varphi \rrbracket \neq \emptyset$ and hence we can apply part 2. of Theorem 3: there is a transition $(\mathbf{1}, Z) \xrightarrow{\lambda(e)} (\mathbf{1}', Z')$ and $\text{GS}(\omega(K \cup \{e\})) = (\mathbf{1}', Z')$. As $\text{GS}(K \cup \{e\}) = \text{GS}(\omega(K \cup \{e\}))$ the result follows.

Proving $\llbracket \Psi(\{\perp\}) \rrbracket \neq \emptyset$ implies $\llbracket \Phi_{\omega(\{\perp\})} \rrbracket \neq \emptyset$ and $\text{GS}(\{\perp\}) = (\mathbf{1}_0, Z_0)$ is easy and this completes the proof. \square

This proves that $\text{EBP}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$ contains correct information the reachable states and the firing of transitions. Assume we keep only the events of $\text{EBP}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$ that are in $\text{TBP}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$ *i.e.* when building $\text{EBP}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$ we add an event e only if $\omega(e)$ is not a cut-off events. Then the EBP obtained is finite, correct and complete and is a complete finite prefix for $\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$.

Theorem 5. *Let (E, P) be the restriction of $\text{EBP}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$ to the set of events e s.t. $\omega(e)$ is in $\text{PREFIX}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$. Then (E, P) is a complete finite prefix for $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$.*

Proof. The fact that (E, P) is correct and finite follows from Theorem 4 and finiteness of $\text{ENABLE}(e)$ for each e in $\text{PREFIX}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$. Completeness is ensured because as well because $\text{ENABLE}(e)$ a safe representative for each possible cut that can enable e . \square

Applying the (implicit) algorithm defined by Definition 14 and adding only non cut-off events give the extended complete finite prefix of Fig. 2. We let $\text{EPREF}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$ be the extended complete finite prefix of $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$.

4.5 Solution to Problem 2

Let (E, P) be $\text{EPREF}((\mathcal{A}_1 | \dots | \mathcal{A}_n)_I)$. To complete the construction and provide a solution to Problem 2, we define the constraint $\Gamma(e)$ associated with an event e of (E, P) by: $\Gamma(e) = \Psi(\lceil e \rceil)_{\lceil e \rceil \cap E}$. The branching process obtained this way is a *reduced* branching process with only constraints on events. For the network of timed automata of Fig. 1, the reduced branching process is given on Fig. 6.

5 Conclusion

In this paper we have defined a model, *symbolic extended nets*, to define the concurrent semantics of timed systems. We have also proved that each NTA admits a *finite complete prefix* which is a symbolic extended net, and we have given an algorithm to compute such a prefix. Other interesting results are:

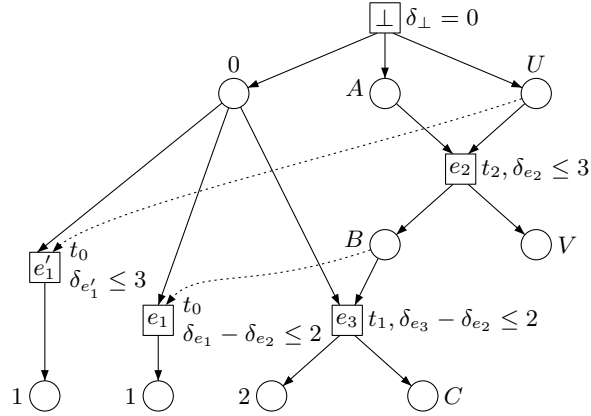


Figure 6. Reduced Extended symbolic unfolding for the example of Fig. 1

- there is no unique complete finite prefix for a NTA but rather a set of complete finite prefixes;
- building a *small* (optimal) complete finite prefix is very expensive as it requires the computation of information spread across the network;
- we have pointed out the difficulties arising in the construction of such a prefix, namely the need for *safe co-sets* and the *Time Lock Freedom* aspects.

Our future work will consist in:

1. define heuristics to determine when an event can be added to a prefix of an unfolding; this means having an efficient way of computing safe representatives, which are no more guaranteed to be minimal.
2. when step 1 is more developed, we can define algorithms to check properties of the NTA using the unfolding and assess the efficiency of these algorithms;
3. we have given a *symbolic* partial order semantics for NTA but have not defined directly a *timed* partial order semantics. Such a definition could be interesting to have a better understanding of the interaction between time and concurrency, and maybe helpful to improve the efficiency of Step 1. This is a key issue before we can use this framework on real-case studies.

References

1. Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim G. Larsen. The power of reachability testing for timed automata. *Theoretical Computer Science (TCS)*, 300(1–3):411–475, 2003.
2. Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science (TCS)*, 126(2):183–235, 1994.

3. Tobias Amnell, Gerd Behrmann, Johan Bengtsson, Pedro R. D'Argenio, Alexandre David, Angskar Fehnker, Thomas Hune, Bertrand Jeannet, Kim G. Larsen, Oliver Möller, Paul Pettersson, Carsten Weise, and Wang Yi. UPPAAL – now, next, and future. In *Proc. Modelling and Verification of Parallel Processes (MOVEP2k)*, volume 2067 of *Lecture Notes in Computer Science*, pages 99–124. Springer, 2001.
4. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.
5. Gerd Behrmann, Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient timed reachability analysis using clock difference diagrams. In *Proc. 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353. Springer, 1999.
6. Béatrice Bérard, Franck Cassez, Serge Haddad, Olivier H. Roux, and Didier Lime. Comparison of the Expressiveness of Timed Automata and Time Petri Nets. In Paul Pettersson and Wang Yi, editors, *Proceedings of the third International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 211–225, Uppsala, Sweden, September 2005. Springer.
7. Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. Software Eng.*, 17(3):259–273, 1991.
8. Bernard Berthomieu, Pierre-Olivier Ribert, and François Vernadat. L'outil tina – construction d'espaces d'états abstraits pour les réseaux de Petri et réseaux temporels. In *Actes 4ième Colloque sur la Modélisation des Systèmes Réactifs (MSR'2003)*, pages 295–310. Hermès, 2003.
9. Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.
10. Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. KRONOS: a model-checking tool for real-time systems. In *Proc. 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.
11. Marius Bozga, Oded Maler, Amir Pnueli, and Sergio Yovine. Some progress in the symbolic verification of timed automata. In *Proc. 9th International Conference on Computer-Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 1997.
12. Franck Cassez and Olivier H. Roux. Structural translation from time petri nets to timed automata. *Journal of Systems and Software*, 2006. forthcoming.
13. Thomas Chatain and Claude Jard. Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In *ICATPN*, volume 4024 of *LNCS*, pages 125–145, june 2006.
14. Javier Esparza and Stefan Römer. An unfolding algorithm for synchronous products of transition systems. In *CONCUR*, volume 1664 of *LNCS*, pages 2–20. Springer, 1999.
15. Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.
16. Hans Fleischhack and Christian Stehno. Computing a finite prefix of a time Petri net. In *ICATPN*, pages 163–181, 2002.
17. Guillaume Gardey, Didier Lime, Morgan Magnin, and Olivier (H.) Roux. Romeo: a tool for analyzing time petri nets. In *Proc. 17th International Conference on*

- Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK, July 2005. Springer–Verlag.
18. Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th IEEE Annual Symposium on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.
 19. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTECH: A model-checker for hybrid systems. *Journal on Software Tools for Technology Transfer (STTT)*, 1(1–2):110–122, 1997.
 20. J. Bengtsson, B. Jonsson, J. Lilius, W. Yi. Partial order reductions for timed systems. In *CONCUR 99*, volume 1466 of *LNCS*, pages 485–500, 1999.
 21. François Laroussinie and Kim G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint International Conference on Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic, 1998.
 22. Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 14–24. IEEE Computer Society Press, 1997.
 23. Denis Lugiez, Peter Niebert, and Sarah Zennou. A partial order semantics approach to the clock explosion problem of timed automata. In *Proc. 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2004)*, volume 2988 of *Lecture Notes in Computer Science*, pages 296–311. Springer, 2004.
 24. Kenneth L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.
 25. P.M. Merlin and D.J. Farber. Recoverability of communication protocols – implications of a theoretical study. *IEEE Transactions on Communications*, 24, 1976.
 26. M. Minea. Partial order reduction for model checking of timed automata. In *CONCUR 99*, volume 1664 of *LNCS*, pages 431–446, 1999.
 27. T. Aura and J. Lilius. A causal semantics for time petri nets. *Theoretical Computer Science*, 1–2(243):409–447, 2000.
 28. T. Yoneda, B-H. Schlingloff. Efficient verification of parallel real-time systems. *Formal Methods in System Design*, 2(11):187–215, 1997.
 29. Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.
 30. A. Valmari. Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets*, volume 483 of *LNCS*, pages 491–515, 1989.
 31. W. Belluomini, C. J. Myers. Verification of timed systems using posets. In *CAV 98*, volume 1427 of *LNCS*, pages 403–415, 1998.

A Time Lock Freedom

Assume we have the network of two automata defined by Fig. 7. As they are independent, it seems quite strange that a dependency on the other automata should be needed in the unfolding, because unfoldings are supposed to take advantage of concurrent events. Considering the network of Fig. 8, the reason why such read arcs would be necessary becomes clearer: the two automata are not really independent because one of them can cause a time lock. Consequently,

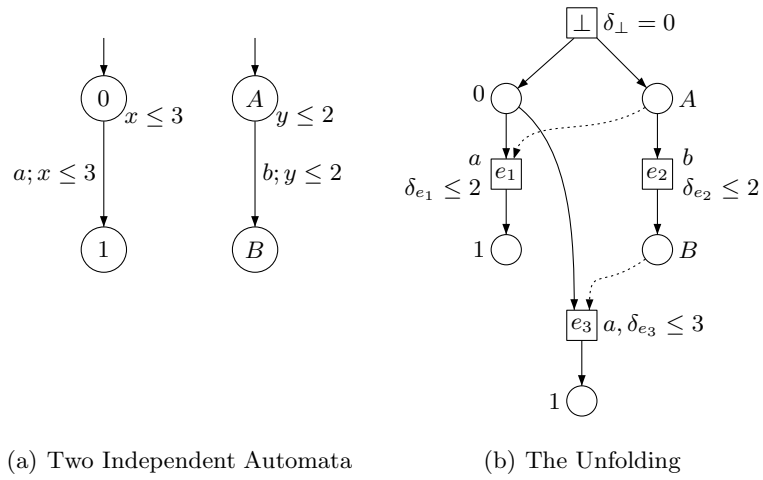


Figure 7. A Network of two Independent Timed Automata

we need to take into account the maximum time that can elapse in the NTA and this implies for the automaton (a) of Fig. 8 to fire e_1 before time 2 which is given by the place labelled A in the unfolding. This is not necessary in the NTA of example 7 because none of the two automata can prevent time from elapsing. This is why the time lock freedom assumption is crucial otherwise a lot of dependencies would appear, most of them are not needed.

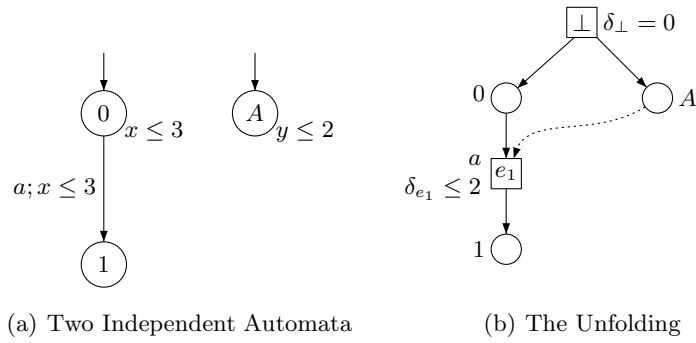


Figure 8. An Example with a Time Lock