Symbolic Unfoldings For Networks of Timed Automata

Franck Cassez[1], Thomas Chatain[2] and Claude Jard[3]

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

[2] IRISA/INRIA, Campus be 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 *extended symbolic nets*. Symbolic nets are standard occurrence nets extended with *read arcs* and *symbolic constraints* on places and transitions. We prove that there is a *complete finite prefix* for any NTA that contains at least the information of the simulation graph of the NTA but keep explicit the notions of concurrency and causality of the network.

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. [31]) that take advantage of the *independence* between actions, and to the *unfolding* based methods [16,25] 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) [2,19], and *time* Petri nets (TPN) [26]. There are a number of theoretical results about NTA and TPN and efficient tools to analyze them have been developped ([3,10,22,20,8,18]). 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 [21,27], 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 [24] (a generalization of [32]), 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 no 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 [29], 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 developped 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 [28] 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 [17] 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 [28] 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 transition[1] in a NTA. As stated in [28] 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 [14], we propose in this paper a similar notion for NTA, but we cannot directly apply the framework of [14]. Indeed TA and TPN have different expressive powers ([6,13]) and as stated earlier NTA do not have the nice *urgency* features that TPN have.

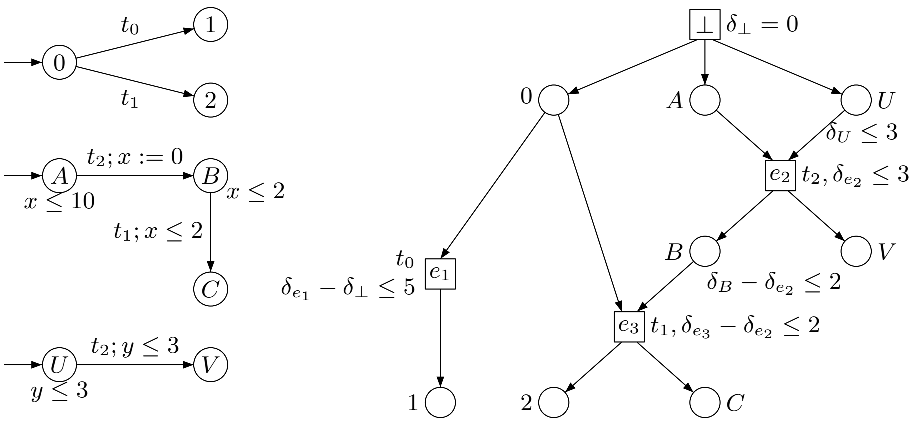
Up to our knowledge, this is the first attempt to equip NTA with a concurrent semantics, which can be finitely represented by a prefix of an unfolding. In this paper we answer the following questions:

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 [14];
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.

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

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 [14] 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), a finite complete prefix[2] is given on Fig. 1(b). 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. For



t_0
t_1
0
1
2
$t_2; x := 0$
A
B
$x \leq 10$
$x \leq 2$
$t_1; x \leq 2$
C
$t_2; y \leq 3$
U
V
$y \leq 3$
$\perp$ $\delta_\perp = 0$
0
A
U
$\delta_U \leq 3$
e_2 $t_2, \delta_{e_2} \leq 3$
B
V
$\delta_B - \delta_{e_2} \leq 2$
t_0
e_1
$\delta_{e_1} - \delta_\perp \leq 5$
e_3 $t_1, \delta_{e_3} - \delta_{e_2} \leq 2$
1
2
C

(a) The NTA $\mathcal{N}_1$

(b) Symbolic unfolding for the network $\mathcal{N}_1$

Figure 1. A NTA and its Symbolic Unfolding

example, $\{t_1\}$ is not a feasible set of events in the network $\mathcal{N}_1$, because t_1 must be preceded by t_2. And this appears in the unfolding as event e_3 (labelled by t_1) must be preceeded by e_2 (labelled by t_2). 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* (or *local configuration*) of e. Note that by definition a *configuration*

[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.

contains the causal past of each of the event. A *complete* prefix is an unfolding that satisfies property (P): a set of events is feasible in the NTA iff it is a configuration of the unfolding[3]. This property of unfoldings is the key point in the untimed case and allows one to do model-checking on the complete finite prefix. This unfolding can also be used for *fault diagnosis* purposes which is a very important application area.

In the case of networks of *timed automata*, we deal with *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 build 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 the constraint $g(e_1) \overset{def}{=} \delta_{e_1} - \delta_\perp \leq 5$, where δ_e is the variable that represents the date of occurrence of e. A set of timed events $\{(e_1, d_1), \cdots, (e_k, d_k)\}$ is a *timed configuration* if $\{e_1, e_2, \cdots, e_k\}$ is a configuration and the constraint $g(e_1) \wedge \cdots \wedge g(e_k)$ is satisfied when replacing each δ_{e_i} by d_i. For example $\{(\perp, 0), (e_1, 4)\}$ is a timed configuration with $g(\perp) \overset{def}{=} \delta_\perp = 0$. Thus the property we would like to have for symbolic unfoldings is (P'): $\{(e_1, d_1), \cdots, (e_k, d_k)\}$ is feasible iff it is a timed configuration.

In the untimed case, one can check that an event is fireable in the unfolding using only the causal past of the event. We want this property to hold for the timed unfoldings as well and then a formula associated with an event e should only involve variables that are associated with events in the causal past of e (the local configuration of e). Now assume we want to decide whether $\{(\perp, 0), (e_1, d_1), (e_2, d_2)\}$ is a timed configuration. It is actually if $d_1 - d_2 \leq 2$. 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 with 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 show 1) how to build an unfolding that contains enough information so that (P') holds; 2) how to build a finite and complete prefix of the unfolding satisfying (P').

Organization of the Paper. The paper is organized as follows. Section 2 presents the model of NTA and its usual sequential semantics. Section 3 gives a concurrent semantics for NTA in terms of *symbolic branching processes* (SBP) and proves the existence of complete finite prefixes. The SBP is a first step towards a complete finite prefix having property (P'). In section 4, we show how to build an *extended* SBP, using *read-arcs*, which is a complete finite prefix satisfying property (P'). Section 5 gives a summary of the paper and directions for future work. The proofs of the theorems are omitted and can be found in the extended version of the paper [12].

2 Networks of timed automata

Notations . Let $X = \{x_1, \cdots, x_n\}$ be a finite set of *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

[3] Actually we should write "it is a labeling" of a configuration of the unfolding.

X' and is defined by $\nu_{|X'}(x) = \nu(x)$ for $x \in X'$. We denote **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 $\nu \in \mathbb{R}^X_{\geq 0}$, we use $\varphi[x/\nu(x)]$ for φ where x is replaced by $\nu(x)$. we denote $\varphi(\nu) \in \{\mathsf{tt}, \mathsf{ff}\}$ the truth value of $\varphi[x/\nu(x)]$. We let $[\![\varphi]\!] = \{\nu \in \mathbb{R}_{\geq 0} \,|\, \varphi(\nu) = \mathsf{tt}\}$. A subset Z of $\mathbb{R}^X_{\geq 0}$ is a zone if $Z = [\![\varphi_Z]\!]$ 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, $Z^{\nearrow} = \{v + \delta \,|\, v \in Z, \delta \in \mathbb{R}_{\geq 0}\}$ and the *R-reset* operator, $Z[R] = \{v \,|\, \exists v' \in Z \text{ s.t. } v = v'[R]\}$. Both $Z^{\nearrow}$ and $Z[R]$ are zones if Z is a zone. A *simple substitution* s is a mapping $s : X \to \mathsf{diff}(Y)$, where $\mathsf{diff}(Y) = \{y - y' \,|\, y, y' \in Y\}$ 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, $\varphi[s] \in \mathcal{C}(Y)$.

Timed Automata. *Timed automata* were introduced in [2] to model systems which combine *discrete* and *continuous* evolutions.

Definition 1. *A* timed automaton $\mathcal{A}$ *is a tuple* $(L, \ell_0, \Sigma, X, T, \textsc{Inv})$ *where:* L *is a finite set of* locations*;* ℓ_0 *is the* initial *location;* Σ *is a finite set of* discrete *actions;* $X = \{x_1, \cdots, x_n\}$ *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* ℓ'*, labeled by* a*, with the guard* g *and the reset set* $R \subseteq X$*; we write* $\textsc{src}(t) = \ell$, $\textsc{tgt}(t) = \ell'$, $\textsc{Guard}(t) = g$, $\lambda(t) = a$ *and* $\textsc{Reset}(t) = R$. $\textsc{Inv} \in \mathcal{C}_r(X)^L$ *assigns an* invariant *to any location. We require that* $\textsc{Inv}$ *be a conjunction 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 $(\ell, v) \in L \times \mathbb{R}^X_{\geq 0}$. A timed automaton is *bounded* if there exists a constant $k \in \mathbb{N}$ s.t. for each $\ell \in L$, $\textsc{Inv}(\ell) \subseteq [\![0 \leq x_1 \leq k \wedge \cdots \wedge 0 \leq x_n \leq k]\!]$. Examples of timed automata are given in Fig. 1(a). In the sequel we require that for any valuation v and any transition $t = (\ell, g, a, R, \ell')$, $g(v) \implies \textsc{Inv}(\ell')(v[R])$. An automaton satisfying this property is said to be *simple*. This assumption simplifies condition i) of Def. 2 below, and also the proofs of the main theorems, because we can decide whether t can be fired using $\textsc{Guard}(t)$. 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.

Definition 2. *The semantics of a timed automaton* $\mathcal{A} = (L, \ell_0, \Sigma, X, T, Inv)$ *is a labeled timed transition system* $S_{\mathcal{A}} = (Q, q_0, T \cup \mathbb{R}_{\geq 0}, \to)$ *with* $Q = L \times (\mathbb{R}_{\leq 0})^X$, $q_0 = (\ell_0, \mathbf{0})$ *is the initial state and* $\to$ *consists of the discrete and continuous transition relations: i) the discrete transition relation is defined for all* $t \in T$ *by:* $(\ell, v) \xrightarrow{t} (\ell', v') \iff \exists\, t = (\ell, g, a, R, \ell') \in T$ *s.t.* $g(v) = \mathsf{tt}$, $v' = v[R \mapsto 0]$ *(we do not need to add* $Inv(\ell')(v') = \mathsf{tt}$ *because we assume our automaton is simple); ii) the continuous* *transition relation is defined for all $\delta \in \mathbb{R}_{\geq 0}$ by: $(\ell, v) \xrightarrow{\delta} (\ell', v')$ iff $\ell = \ell'$, $v' = v + \delta$ and $\forall\, 0 \leq \delta' \leq \delta,\ Inv(\ell)(v + \delta') = \mathit{tt}$. 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*[4]*. 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 $\mathcal{A}$. A timed word $w \in (T \times \mathbb{R}_{\geq 0})^*$ is* accepted *by $\mathcal{A}$ is there is a run $\rho \in [\![\mathcal{A}]\!]$ s.t. the trace of ρ is w.*

The analysis [1,5,9,30,11] 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 $\mathbb{R}^X_{\geq 0}$.

Definition 3. *The* simulation graph $SG(\mathcal{A})$ *of a timed automaton $\mathcal{A}$ is given by: i) the set of states is the set of symbolic states of the form (ℓ, Z) where Z is a zone; ii) the initial state is (ℓ_0, Z_0) with $Z_0 = \mathbf{0}^{\nearrow} \cap\ [\![\text{INV}(\ell_0)]\!]$; iii) $(\ell, Z) \xrightarrow{a} (\ell', Z')$ 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 $Z' = \big((Z \cap\ [\![g]\!])[R]\big)^{\nearrow} \cap\ [\![\text{INV}(\ell')]\!]$.*

We assume that the timed automata are bounded *i.e.* in each location ℓ, $Inv(\ell)$ is bounded[5]. In this case the number of zones of the simulation graph is finite [23,9].

Product of Timed Automata. We use the classical composition notion based on a *synchronization function* à la Arnold-Nivat. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be n timed automata with $\mathcal{A}_i = (L_i, l_{i,0}, \Sigma_i, X_i, T_i, Inv_i)$. 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 $\Sigma_1^\varepsilon \times \Sigma_2^\varepsilon \cdots \times \Sigma_n^\varepsilon \setminus (\varepsilon, \cdots, \varepsilon)$. The (synchronization) vectors of a synchronization constraint I indicate which actions synchronize. If $z = (a_1, \cdots, a_n) \in I$ we write $z[j]$ for the j-th component a_j. For $(t_1, \cdots, t_n) \in T_1^\varepsilon \times \cdots T_n^\varepsilon$ we write $\lambda(t_1, \cdots, t_n) = (\lambda_1(t_1), \cdots, \lambda_n(t_n))$ with $\lambda_i(\varepsilon) = \varepsilon$. For $z \in I$, we define $\lambda^{-1}(z) = \{t \in T_1^\varepsilon \times \cdots T_n^\varepsilon \,|\, \lambda(t) = z\}$ 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: $\text{SRC}^*(t) = \{l \in \text{SRC}(t[i]) \,|\, t[i] \neq \varepsilon\}$, $\text{TGT}^*(t) = \{l \in \text{TGT}(t[i]) \,|\, t[i] \neq \varepsilon\}$, $\text{RESET}(t) = \{x \,|\, x \in \text{RESET}(t[i]) \textit{ and } t[i] \neq \varepsilon\}$, $\text{GUARD}(t) = \wedge_{t[i] \neq \varepsilon} \text{GUARD}(t[i])$.

Definition 4. *The synchronous product $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_I$ is the timed automaton $\mathcal{B} = (L, \mathbf{l}_0, \Sigma, X, T, \text{INV})$ defined by: $L = L_1 \times \cdots \times L_n$, $\mathbf{l}_0 = (\ell_{1,0}, \cdots, \ell_{n,0})$, $\Sigma = \Sigma_1 \times \cdots \times \Sigma_n$, $X = \cup_{i=1}^n X_i$; $(\mathbf{l}, g, a, R, \mathbf{l}') \in T$ 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 $R = \text{RESET}(t)$ and $\text{INV}(\mathbf{l}) = \wedge_{i=1}^n \text{INV}_i(\ell_i)$ if $\mathbf{l} = (\ell_1, \cdots, \ell_n)$.*

This definition implies that if each $\mathcal{A}_i$ is bounded (resp. simple) then the product is bounded (resp. simple).

[4] In our definition runs are labeled by transitions.

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

3 Symbolic Unfolding for Network of Timed Automata

In this section we define the symbolic semantics of a NTA in terms of *symbolic branching processes*. Those processes contain timing constraints both on places and events. The definitions of *occurrence nets*, *branching processes*, together with the notions of *co-sets*, *cuts*, *conflict* are taken from [15] and recalled in appendix A. We denote $(E, P, {}^\bullet(), ()^\bullet)$ an occurrence net. Given a set B we denote $\delta(B)$ the set of (fresh) variables $\{\delta_b \mid b \in B\}$. $\lceil x \rceil$ denotes the causal past of x.

Definition 5. *A* symbolic occurrence net $\mathcal{T}$ *is a tuple* $(E, P, {}^\bullet(), ()^\bullet, \gamma)$ *where* $(E, P, {}^\bullet(), ()^\bullet)$ *is an occurrence net, and* $\gamma : E \cup P \to \mathcal{C}(X)$ *with* $X = \delta(E \cup P)$. *We require that i) for each* $x \in E \cup P$, $\gamma(x)$ *contains only variables in* $\delta(\lceil x \rceil)$*, and ii)* $\gamma(\bot) \overset{def}{=} (\delta_\bot = 0)$. *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. 1(b).

Definition 6. (M, Φ) *is a* symbolic co-set *of* $\mathcal{T}$ *if: 1)* M *is a co-set of the underlying net, 2)* $\Phi = \Phi_1(M) \wedge \Phi_2(M) \wedge \Phi_3(M) \wedge \Phi_4(M)$ *with:*

$$\Phi_1(M) = \bigwedge_{x \in \lceil M \rceil} \gamma(x) \quad (1) \qquad \Phi_3(M) = \bigwedge_{p \in M} \left(\delta_{{}^\bullet p} \le \delta_p\right) \quad (3)$$

$$\Phi_2(M) = \bigwedge_{e \in \lceil M \rceil \cap E} \left(\wedge_{p \in {}^\bullet e} \delta_p = \delta_e\right) \quad (2) \qquad \Phi_4(M) = \Big(\bigwedge_{p, p' \in M} \delta_p = \delta_{p'}\Big) \quad (4)$$

(M, Φ) *is a symbolic* cut *is* M *is a cut of the underlying net.*

The meaning of formula (2) is that the more recent date δ_p at which a token was in p is the time at which an event removed a token in p. (3) imposes that if a token is in p and p is in a co-set, the current time in p which is δ_p is larger than the date of occurrence of the event that put a token in p. Finally (4) requires that all the places in the co-set have reached the same global time. The reason why we need to use variables associated with places is because there is no urgency in NTA. Notice that the formula Φ of a symbolic co-set is entirely determined by the co-set M and unique; we denote it by Φ_M. Let M be a co-set $\nu : \delta(\lceil M \rceil) \to \mathbb{R}_{\ge 0}$. (M, ν) is a *timed co-set* if $\nu \in [\![\Phi_M]\!]$. Consequently $(M, [\![\Phi_M]\!])$ is the set of timed co-sets associated with M. (M, ν) is a *timed cut* if M is a cut and (M, ν) is a timed co-set.

Let $(\mathcal{A}_1 | \dots | \mathcal{A}_n)_I$ be a synchronous product of TA. 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: i) $\bot \in \mathcal{E}$, ii) if $e \in \mathcal{E}$ and $s \in L$ then $(e, s) \in \mathcal{P}$, iii) if $S \subseteq \mathcal{P}$ and $t \in I$ then $(S, t) \in \mathcal{E}$. On those two sets we define the mappings ${}^\bullet()$, $()^\bullet$: for $\mathcal{E}$, ${}^\bullet\bot = \emptyset$, 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 ans 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 (i–ii) of Def. 5. We use the following notations:

– for $e = (S, t) \in \mathcal{E} \setminus \{\bot\}$, $\lambda(e) = t$; we say that e is an i-event if $t[i] \neq \varepsilon$;

– for $(e,s) \in \mathcal{P}$, $\text{loc}(e,s) = s$, $\text{Loc}(Y) = \cup_{y \in Y} \text{loc}(y)$. (e,s) is an i-place if $s \in L_i$;
– $\textsc{Reset}(e) = \textsc{Reset}(\lambda(e))$, and by convention $\textsc{Reset}(\bot) = X$.
– if $x \in X$ and K is a configuration of a SBP, the *event of last reset of* x *in* K is $Z_K^{:=0}(x) = \min\{e \in K \mid x \in \textsc{Reset}(e)\}$. A minimal element always exist as $\bot$ 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 7. *The set of* symbolic finite branching processes *of a network* $(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$ *is defined inductively as follows:*

– $(\{\bot\}, \{(\bot, l_{1,0}), (\bot, l_{2,0}), \cdots, (\bot, l_{n,0})\}, \gamma)$ *with* $\gamma(\bot) \overset{\text{def}}{=} \delta_\bot = 0$, $\gamma(\bot, l_{i,0}) \overset{\text{def}}{=} \textsc{Inv}(l_{i,0})[s_i]$, *with the simple substitution* s_i *defined by* $s_i(x) = \delta_{(\bot, l_{i,0})}$ *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. Loc*$(C) = \textsc{src}^*(t)$, *then*
$$(E \cup \{e\}, P \cup \{(e,s) \mid s \in \textsc{tgt}^*(t)\}, \gamma')$$
is a SBP of $(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$, *with* $e = (C,t)$, $\gamma'_{|E \cup P} = \gamma$, $\gamma(e) = \textsc{Guard}(t)[\eta]$, $\eta : x \mapsto \delta_e - \delta_{Z^{:=0}_{\lceil C \rceil}(x)}$, $\gamma(e,s) = Inv(s)[\eta']$, $\eta' : x \mapsto \delta_{(e,s)} - \delta_{Z^{:=0}_{\lceil e \rceil}(x)}$ *(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 (i–ii) of Definition 5. An example of a symbolic branching process is given on Fig. 1(b)[6]. We can 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 $(E_1 \cup E_2, P_1 \cup P_2, \gamma)$, and[7] $\gamma(x) = \gamma_i(x)$ 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*, $\textsc{tbp}(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$, 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 [15]. 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 carry over from the results of ([15]).

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 one* i*-place for each* $1 \le i \le n$.

[6] 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 $\gamma(e)$.
[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 $P_1 \cup P_2$.

This enables us to associate with each configuration K, a symbolic state. Let $\textsc{Cut}(K)$ 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 $\textsc{Cut}(K)$: $\mathbf{l}$ is the "vector" representation of the set $\mathrm{Loc}(\textsc{Cut}(K))$. Now consider the formula $f = \Phi_{\textsc{Cut}(K)}$ defined in Definition 6. This formula is in $\mathcal{C}(\delta(\lceil \textsc{Cut}(K)\rceil))$. Assume $\nu \in [\![f]\!]$. Define $v_\nu \in \mathbb{R}^X_{\geq 0}$ by[8]: $v_\nu(x) = \nu(\textsc{Cut}(K)) - \nu(\delta_{Z^{:=0}_{\lceil K\rceil}(x)})$. We let $Z_K = \{v_\nu \,|\, \nu \in [\![f]\!]\}$. The symbolic global state associated with K is $\mathrm{GS}(K) = (\mathrm{Loc}(\textsc{Cut}(K)), Z_K)$. Note that Z_K can be the empty set. $\textsc{tbp}(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$ contains correct and complete information w.r.t. the simulation graph of $(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$:

Theorem 1. *If K is a configuration of $\textsc{tbp}(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$ s.t. $\mathrm{GS}(K) \neq \emptyset$ then a) $\mathrm{GS}(K) = (\mathbf{l}, Z)$ for some $(\mathbf{l}, Z)$ reachable in $SG((\mathcal{A}_1|\dots|\mathcal{A}_n)_I)$, and b) if $K \cup \{e\}$ is a configuration and $[\![\Phi_{\textsc{Cut}(K)} \wedge \gamma(e) \bigwedge(\wedge_{p\in {}^\bullet e}\delta_p = \delta_e)]\!] \neq \emptyset$ then $(\mathbf{l}, Z) \xrightarrow{\lambda(e)} (\mathbf{l}', Z')$ and $\mathrm{GS}(K \cup \{e\}) = (\mathbf{l}', Z')$.*

As a consequence of Theorem 1, we obtain that each Z_K is a zone for a configuration K. Moreover this zone can be computed effectively (see [12]). A corollary of Theorem 1 is:

Corollary 1. *Let $K = \{e_1, \cdots, e_j\}$ be a configuration and $\nu : \lceil \textsc{Cut}(K)\rceil \to \mathbb{R}_{\geq 0}$ s.t. $\nu \in [\![\Phi_{\textsc{Cut}(K)}]\!] \neq \emptyset$. Then there exists a one-to-one mapping $f : [1..j] \to [1..j]$ s.t. $(\lambda(e_{f(1)}), \nu(e_{f(1)}) \cdots (\lambda(e_{f(j)}), \nu(e_{f(j)})$ is a timed word accepted by $(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$.*

Corollary 1 states that if the formula associated with $\textsc{Cut}(K)$ is satisfiable then there is a way of ordering the events in K such that they produce a timed word of the NTA.

Theorem 2. *Let $(\mathbf{l}, Z)$ 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 $\textsc{tbp}(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$ s.t.: a) $\mathrm{GS}(K) = (\mathbf{l}, Z)$ and, b)if $(\mathbf{l}, Z) \xrightarrow{t} (\mathbf{l}', Z')$ there is a configuration $K \cup \{e\}$, with $\lambda(e) = t$ and $\mathrm{GS}(K \cup \{e\}) = (\mathbf{l}', Z')$.*

Corollary 2. *Let $(w_1, d_1)(w_2, d_2) \cdots (w_k, d_k)$ be a timed word accepted by $(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$. Then, there exists a configuration $K = \{e_1, \cdots, e_k\}$ of $\textsc{tbp}(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$ and $\nu : \lceil \textsc{Cut}(K)\rceil \to \mathbb{R}_{\geq 0}$, $\nu \in [\![\Phi_{\textsc{Cut}(K)}]\!]$, s.t. 1) $\lambda(e_i) = w_i$ and 2) $\nu(\delta_{e_i}) = d_i$.*

If a TBP $\mathcal{T}$ satisfies the conditions of Theorem 2, we say that $\mathcal{T}$ is *complete*. Theorem 1, corresponds to a *correctness* property. For network of finite automata, complete (and correct) finite branching processes exist, and are called *complete finite prefixes* [25,15]. 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

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

as the simulation graph, we can re-use the algorithms of [15] and the notion of *cut-off* events and *adequate orders*. Adequate orders are defined in [15] and we refer the reader to this article for a comprehensive definition and list of the properties of these orders.

As a consequence symbolic complete prefixes exist for network of timed automata and can be computed using the efficient algorithms of [15]. We let $\textsc{Pref}((\mathcal{A}_1|\dots|\mathcal{A}_n)_I)$ be the complete symbolic finite prefix obtained from $(\mathcal{A}_1|\dots|\mathcal{A}_n)_I$. The example of Fig. 4 in appendix C, page 18 has an infinite unfolding (synchronization is on common labels). A complete finite prefix is given on Fig. 4.(a), page 18.

The unfolding of Fig. 1(b) is trivially a complete finite prefix because it is a finite unfolding. As pointed out in the introduction, our SBP does not always satisfy property (P'). In the next section we refine the SBP to obtain an *extended* SBP that satisfies property (P').

4 Extended Finite Complete 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. 1(b). If e_2 has not fired, 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 that the date δ_{e_1} at which e_1 fires satisfies $\delta_{e_1} \leq 3$. 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.* to fire we must have $\delta_B = \delta_{e_1}$ and the constraint on the date at which a token can be in B which is $\delta_B - \delta_{e_2} \leq 2$. This implies $\delta_{e_1} - \delta_{e_2} \leq 2$. Thus the timing constraints associated with e_1 are not the same in the cuts $(0, A, U)$ and $(0, B, V)$ although they are both cuts that contain ${}^\bullet e_1$.

To encode this timing dependency structurally we can use symbolic occurrence nets with *read arcs*. For instance the symbolic net of Fig. 1(b) can be "transformed" into the symbolic extended net of Fig. 2 (a read arc is a dash line). Read arcs enable us to point to the missing timing information in the net that is needed to ensure an event can fire. This also means that we duplicate the event e_1 into e_1 and e'_1 because the constraints are different depending on whether e_2 has occurred or not. Read arcs enlarge the *causal past* of the events. In the extended occurrence net, the constraint between the dates of occurrence of e_1 and e_2 can be inferred from the past of e_1: indeed, to fire, we must have $\delta_{e_1} = \delta_B$ 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 e_1 and e_2.

Extended nets are nets extended with read arcs. Symbolic extended nets are extended nets with constraints on the places and events. The formal definition of extended symbolic nets together with the new notions of conflict, concurrency and causality are given in appendix B.

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 $e = (S, S', t)$, ${}^\circ e = S'$.

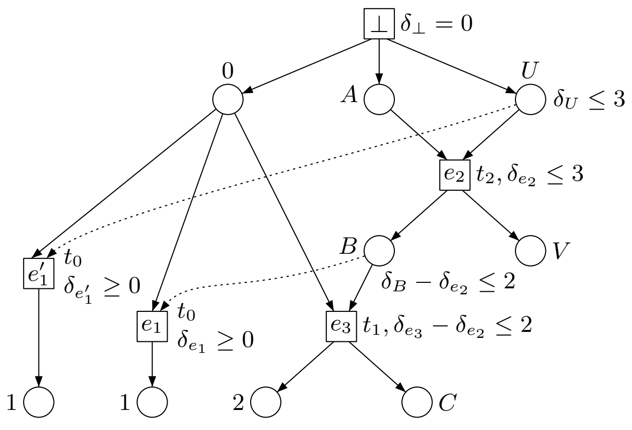


Figure 2. Extended symbolic unfolding for the example of Fig. 1(a)

Time Lock Freedom. The *time lock freedom* assumption, 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' in $\text{TBP}((\mathcal{A}_1|\ldots|\mathcal{A}_n)_I)$, the timing constraint on δ_e obtained by $\varPhi_{(^\bullet e)}$ 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 D.

The NTA of Fig. 1(a) does not satisfy the time lock freedom assumption (automaton 2 can prevent time from elapsing in location B). Nevertheless event e_2 cannot be prevented from firing by the first automaton and thus we can we can safely fire e_2 if the places in $^\bullet e_2$ have a token as the first automata can not 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 $\mathcal{N}$: $\text{Enable}(e) = \{C \mid {^\bullet e} \subseteq C \textit{ and } C \textit{ is a cut of } \mathcal{N}\}$. As a running example we take the prefix $\mathcal{N}_1$ built in Fig. 1(b) and $\delta_\perp$ is always replaced by 0 (zero). For this example the enabling cuts are: $\text{Enable}(e_1) = \{(0, A, U), (0, B, V)\}$, $\text{Enable}(e_2) = \{(0, A, U), (1, A, U)\}$, $\text{Enable}(e_3) = \{(0, B, V)\}$.

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 (this must be at time $\delta \leq 3$), or b) e_2 has fired, and the time elapsed since it has occurred 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 occurrence 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 (see Def. 8 below).

For any cut C, the formula Φ_C (Definition 6, 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 6. 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 any[9] $p \in C$, we obtain a formula over $\delta(\lceil C \rceil \cap E) \cup \{\delta\}$: δ stands for the current global 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, $\Phi^\delta_{(0,A,U)} = \delta \leq 3$, $\Phi^\delta_{(1,A,U)} = \delta_{e_1} \leq 3 \wedge \delta_{e_1} \leq \delta \leq 3$ and $\Phi^\delta_{(0,B,V)} = \delta - \delta_{e_2} \leq 2 \wedge 0 \leq \delta_{e_2} \leq 3 \wedge \delta \geq \delta_{e_2}$. We let $\Theta(e) = \{\Phi^\delta_C \,|\, C \in \textsc{Enable}(e)\}$. In our example, we obtain: $\Theta(e_1) = \{\Phi^\delta_{(0,A,U)}, \Phi^\delta_{(0,B,V)}\}$, $\Theta(e_2) = \{\Phi^\delta_{(0,A,U)}, \Phi^\delta_{(1,A,U)}\}$, $\Theta(e_3) = \{\Phi^\delta_{(0,B,V)}\}$ $\Theta(e)$ 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 $\textsc{Dead}(e)$ be the set defined by: $p \in \textsc{Dead}(e)$ iff for each configuration K, $p \in \lceil \textsc{Cut}(K) \rceil$ 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 8. *A set of places S is a* safe representative *of* e *w.r.t.* e' *if 1)* ${}^\bullet e \subseteq S$ *and 2)* $S \cap \lceil e' \rceil \neq \emptyset$ *or* $S \cap \textsc{Dead}(e') \neq \emptyset$ *and 3)* $\forall C \in \textsc{Enable}(e), S \subseteq C \implies [\![\Phi^\delta_S]\!] = [\![\Phi^\delta_C]\!]$. *$S$ 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. $S \subseteq C$ as $[\![\Phi^\delta_S]\!] = [\![\Phi^\delta_C]\!]$. 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 9. *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 *and 2) for each cut $C \in \textsc{Enable}(e)$, there is some $S \in \mathcal{S}$ s.t. $S \subseteq C$.*

As each cut $C \in \textsc{Enable}(e)$ is a safe representative of itself, there is at least one complete set of safe representatives which is $\textsc{Enable}(e)$. We can state a theorem which is a variant of Theorem 1 using only safe representatives of an event (item b of the theorem):

Theorem 3. *If K is a configuration of* $\textsc{tbp}(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_I$ *s.t.* $\mathrm{GS}(K) \neq \emptyset$ *then a)* $\mathrm{GS}(K) = (\mathbf{l}, Z)$ *for some $(\mathbf{l}, Z)$ reachable in $SG((\mathcal{A}_1 | \ldots | \mathcal{A}_n)_I)$, and b) if $K \cup \{e\}$ is*

[9] 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.

a configuration and S is a safe representative of e s.t. $[\![\Phi_S \wedge \gamma(e) \bigwedge \big(\wedge_{p \in {}^\bullet e} \delta_p = \delta_e\big)]\!] \neq \emptyset$ then $(\mathbf{l}, Z) \xrightarrow{\lambda(e)} (\mathbf{l}', Z')$ and $\mathrm{GS}(K \cup \{e\}) = (\mathbf{l}', Z')$.

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 $(\mathcal{A}_1|\ldots|\mathcal{A}_n)_I$, we add *read-arcs* to the places of a safe representative of e, then $\lceil e \rceil$ (including the read arcs) gives the accurate constraints on the date δ_e at which e can fire.

To build an extended complete finite prefix for a NTA we can proceed as follows: 1) build the symbolic net defined in section 3; this enables us to obtain the safe co-sets for each event; 2) build an extended net by adding an event to the unfolding using safe co-sets instead of simple co-sets. On the example of Fig. 1 this gives the unfolding of Fig. 2:

1. start with places $0, A, U$ and event $\perp$;
2. to add an event labelled t_0 use a safe co-set: we choose $(0, U)$ and add event e_1' with a read arc to U;
3. add e_2 and e_3;
4. now a new safe co-set has appeared: $(0, B)$; we can add an event e_1 labelled by t_0 with a read arc from place B.

This construction can be formally defined (see [12]). The result is a symbolic extended finite complete prefix $\mathrm{EP{\small REF}}((\mathcal{A}_1|\ldots|\mathcal{A}_n)_I)$ that satisfies property (P'). Formally, we define *symbolic configurations*:

Definition 10. *(K, Ψ) is a* symbolic configuration *of $\mathcal{T}$ if: 1) K is a configuration of the underlying net, and 2) $\Psi = \Psi_1(K) \wedge \Psi_2(K)$ where $\Phi_i(M), 1 \leq i \leq 2$ are defined by:*

$$\Psi_1(K) = \bigwedge_{e \in \lceil K \rceil} \gamma(e) \qquad (5) \qquad \textit{and} \qquad \Psi_2(K) = \bigwedge_{e \in K} \big(\wedge_{p \in {}^\bullet e} \delta_p = \delta_e\big) \qquad (6)$$

Notice that $\Psi(K)$ uses only information in the past of K. Let $\nu : K \to \mathbb{R}_{\geq 0}$. (K, ν) is a *timed configuration* if $\nu \in \Psi(K)$. (P') holds because we can now re-write Theorem 3 as follows:

Theorem 4. *If K is a configuration of $\mathrm{EP{\small REF}}((\mathcal{A}_1|\ldots|\mathcal{A}_n)_I)$ s.t. $[\![\Psi(K)]\!] \neq \emptyset$ then: 1) $\mathrm{GS}(K) = (\mathbf{l}, Z)$ for some $(\mathbf{l}, Z)$ reachable in $SG((\mathcal{A}_1|\ldots|\mathcal{A}_n)_I)$, and 2) if $K \cup \{e\}$ is a configuration and $[\![\Psi(K \cup \{e\})]\!] \neq \emptyset$ then $(\mathbf{l}, Z) \xrightarrow{\lambda(e)} (\mathbf{l}', Z')$ and $\mathrm{GS}(K \cup \{e\}) = (\mathbf{l}', Z')$.*

On the example of Fig. 1(a), $\{(\perp, 0), (e_1, \delta_{e_1}), (e_2, \delta_{e_2})\}$ is a timed configuration iff $\delta_{e_1} - \delta_{e_2} \leq 2$ and $\delta_{e_2} \leq 3$. $\{(\perp, 0), (e_1, \delta_{e_1})\}$ is a symbolic configuration iff $\delta_{e_1} \leq 3$.

Minimality for Safe Co-sets. The purpose of unfoldings is to keep explicit the concurrency of events. 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 number of the new dependencies: for instance, if we use $\mathrm{E{\small NABLE}}(e)$ 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 $\mathcal{S} = \{S_1, S_2, \cdots, S_k\}$ for e is *non redundant* if each cut $C \in \text{ENABLE}(e)$ is represented at most once: $\forall C \in \text{ENABLE}(e), C \subseteq S_i$ *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: $\text{SAFE}(e_1) = \{(0, U), (0, B)\}$, $\text{SAFE}(e_2) = \{(A, U)\}$, $\text{SAFE}(e_3) = \{(0, B)\}$. 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 representatives[10]. Also, the minimality criterion we have defined does not give a unique set of complete safe representatives. A consequence is that there is no smallest complete finite prefix for a NTA but rather a set of set of minimal complete finite prefixes.

Checking Validity of Timed Configuration. To complete the construction and provide a solution to the problem of checking whether a timed configuration is valid, we can define the constraint $\Gamma(e)$ associated with an event e by: $\Gamma(e) = \Psi(\lceil e \rceil)_{|\lceil e \rceil \cap E}$. This constraint gathers the constraints of all the past events. 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(a), the reduced branching process is given on Fig. 3.

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:

– 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:

[10] Under the assumption of time lock freedom

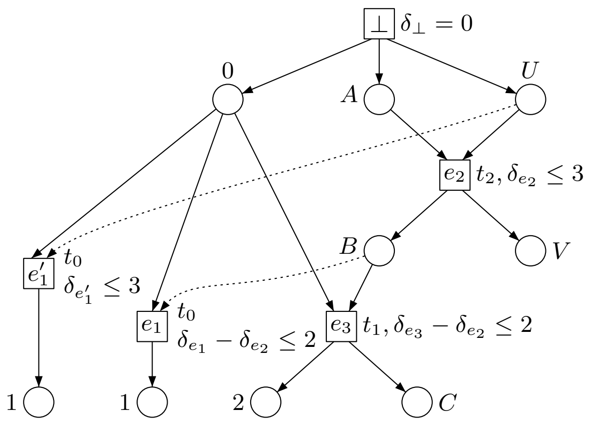

$\perp$ $\delta_\perp = 0$
0
A
U
e_2 $t_2, \delta_{e_2} \le 3$
B
V
e'_1 t_0 $\delta_{e'_1} \le 3$
e_1 t_0 $\delta_{e_1} - \delta_{e_2} \le 2$
e_3 $t_1, \delta_{e_3} - \delta_{e_2} \le 2$
1
1
2
C

Figure 3. Reduced Extended symbolic unfolding for the example of Fig. 1(a)

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 (*MOVEP*2k)*, 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, Thomas Chatain, and Claude Jard. Symbolic Unfoldings for Networks of Timed Automata. Technical Report RI-2006-4, IRCCyN/CNRS, Nantes, May 2006.
13. Franck Cassez and Olivier H. Roux. Structural translation from time petri nets to timed automata. *Journal of Systems and Software*, 2006. forthcoming.
14. 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.
15. 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.
16. Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.
17. Hans Fleischhack and Christian Stehno. Computing a finite prefix of a time Petri net. In *ICATPN*, pages 163–181, 2002.
18. 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 Verfication (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK, July 2005. Springer–Verlag.
19. Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th IEEE Annual Symposim on Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Computer Society Press, 1996.
20. 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.
21. 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.
22. 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.

23. 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.
24. 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.
25. Kenneth L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.
26. P.M. Merlin and D.J. Farber. Recoverability of communication protocols – implications of a theorical study. *IEEE Transactions on Communications*, 24, 1976.
27. M. Minea. Partial order reduction for model checking of timed automata. In *CONCUR 99*, volume 1664 of *LNCS*, pages 431–446, 1999.
28. T. Aura and J. Lilius. A causal semantics for time petri nets. *Theoretical Computer Science*, 1–2(243):409–447, 2000.
29. T. Yoneda, B-H. Schlingloff. Efficient verification of parallel real-time systems. *Formal Methods in System Design*, 2(11):187–215, 1997.
30. Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.
31. A. Valmari. Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets*, volume 483 of *LNCS*, pages 491–515, 1989.
32. W. Belluomini, C. J. Myers. Verification of timed systems using posets. In *CAV 98*, volume 1427 of *LNCS*, pages 403–415, 1998.

A Occurrence Nets

Let x, y be two nodes (place or transitions). 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 $\uparrow J = \big(\cup_{e \in J} e^\bullet\big) \setminus \big(\cup_{e \in J} {}^\bullet e\big)$. For a set $J \subseteq E \cup P$ $\lceil J \rceil = \{e' \in E \cup P \,|\, e' \preceq e \text{ for some } e \in J\}$. 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. 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 $\textsc{Cut}(K)$.

B 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.

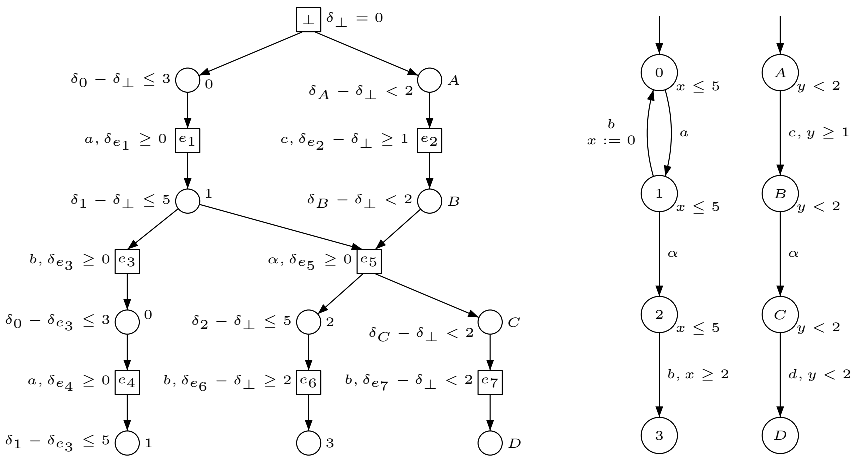
The causality relation is now defined by: $x \rightarrow y$ if $x \in {}^\bullet y \cup {}^\circ y$ or $y \in x^\bullet$. $\preceq$ is the reflexive and transitive closure of $\rightarrow$. The *weak* causality relation $\dashrightarrow$ is given by:

$x \dashrightarrow y$ if either $x \to y$ 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 $\to$.). We let $\trianglelefteq$ the reflexive and transitive closure of $\dashrightarrow$. Two nodes x and y are *weakly causally related* if either $x \trianglelefteq y$ or $y \trianglelefteq x$. 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 $w \trianglelefteq x$ and $u \trianglelefteq y$. 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 5 (using the new $\lceil\cdot\rceil$). The semantics of extended occurrence nets requires that ${}^\bullet e \cup {}^\circ e \subseteq M$ to fire event e.

C Finite Complete Prefix

We use the order $\prec_1$ of [15] to generate this complete finite prefix, and event e_4 is a cut-off event.

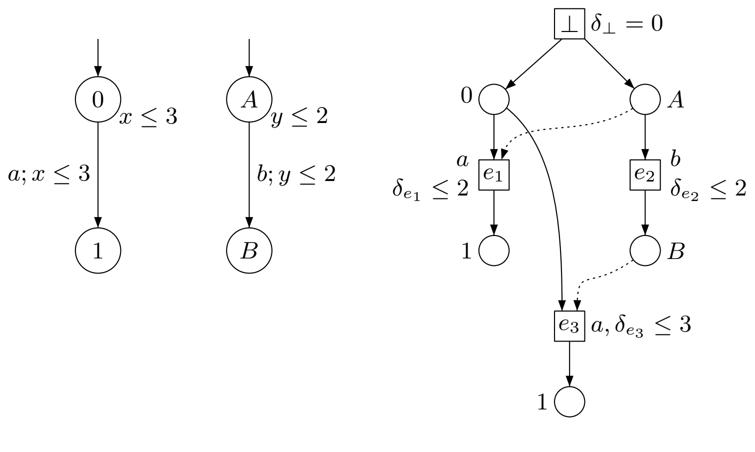


(a) The unfolding (b) The network

Figure 4. A network with a loop

D Time Lock Freedom

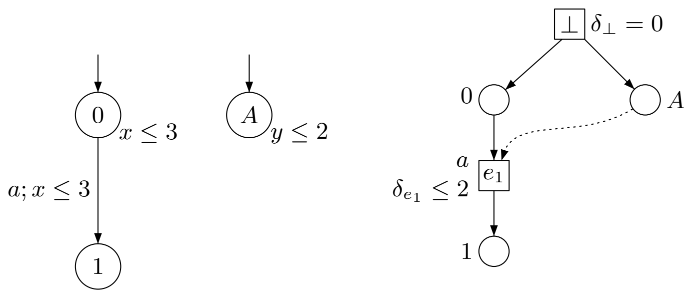
Assume we have the network of two automata defined by Fig. 5. As they are indepen-



(a) Two Independent Automata (b) The Unfolding

Figure 5. A Network of two Independent Timed Automata

dent, 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. 6, 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, we need to take into account the maximum time that can elapse in the NTA and this implies for the automaton (a) of Fig. 6 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 5 because none of the two automata can prevent time from elapsing but we do not know it in advance. This is why the time lock freedom assumption is crucial otherwise a lot of dependencies would appear, most of them are not needed.



(a) Two Independent Automata (b) The Unfolding

Figure 6. An Example with a Time Lock