A Forward Reachability Algorithm for Bounded Timed-Arc Petri Nets*

Alexandre David Lasse Jacobsen Morten Jacobsen Jiří Srba
Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, DK-9220 Aalborg East, Denmark

Timed-arc Petri nets (TAPN) are a well-known time extension of the Petri net model and several translations to networks of timed automata have been proposed for this model. We present a direct, DBM-based algorithm for forward reachability analysis of bounded TAPNs extended with transport arcs, inhibitor arcs and age invariants. We also give a complete proof of its correctness, including reduction techniques based on symmetries and extrapolation. Finally, we augment the algorithm with a novel state-space reduction technique introducing a monotonic ordering on markings and prove its soundness even in the presence of monotonicity-breaking features like age invariants and inhibitor arcs. We implement the algorithm within the model-checker TAPAAL and the experimental results document an encouraging performance compared to verification approaches that translate TAPN models to UPPAAL timed automata.

1 Introduction

Time-dependent models and their formal analysis have attracted a considerable research activity. Notable formalisms include timed automata (TA) [3], time Petri nets (TPN) [18] and timed-arc Petri nets (TAPN) [7]. A comparison of the different modelling formalisms is provided in [23].

We shall focus on the TAPN model where tokens are assigned a nonnegative real number representing their age and input arcs of transitions contain time intervals restricting the usable ages of tokens for transition firing. The state-space of the model is in general infinite in two dimensions: the number of tokens in a marking can be unbounded, and the continuous time aspect induces infinitely many clock valuations. Indeed, the reachability problem for the model is undecidable [21], while coverability remains decidable [2]. Moreover, for modelling purposes additional features like inhibitor/transport arcs and age invariants are needed but they cause the undecidability also of the coverability problem [14].

We restrict our focus to bounded TAPNs where the maximum number of tokens in all reachable markings is fixed. This model is equally expressive to networks of timed automata [22] and efficient translations from TAPN into UPPAAL timed automata [16] have been implemented and employed in the model-checker TAPAAL [9]. The translation approach has though some drawbacks: experimentation with state-space reduction techniques is difficult and the engine does not return error traces when symmetry reduction is enabled.

We therefore design a novel reachability algorithm for extended TAPN that incorporates an efficient extrapolation, symmetry reduction and monotonic inclusion techniques to optimize its performance, while at the same time returning error traces with concrete time delays. We give a complete proof of the algorithm correctness, including all the optimization techniques. We provide an efficient (`C++`), open-source implementation of the algorithm and integrate the new engine into the tool TAPAAL. The experiments confirm a high efficiency of the new reachability algorithm and we document this by two larger case-studies.

*The paper was partially supported by VKR Center of Excellence MT-LAB.

F. Cassez, R. Huuck, G. Klein and B. Schlich (eds.):
Systems Software Verification Conference 2012 (SSV 2012)
EPTCS 102, 2012, pp. 125–140, doi:10.4204/EPTCS.102.12

Related work. Verification techniques for TAPNs include a backward coverability algorithm based on existential zones [2] (notably terminating also for unbounded nets) and a forward reachability algorithm based on region generators presented in [1]. Both algorithms rely on the monotonic behavior of the generated transition systems, however, inhibitor arcs and age invariants break this monotonicity [14] and hence the techniques are not applicable for extended TAPNs. Backward algorithms are generally rather inefficient for on-the-fly state-space exploration and for the employment of state-space reductions while the forward algorithm from [1] is based on a less efficient region construction instead of a zone-based one. The algorithms were implemented in prototype tools with no GUI and are not maintained any more.

There are efficient tools like TINA [6] or Romeo [12] for model-checking Time Petri nets (TPN). The tools are based on abstractions using state-class graphs but even though bounded TPN are essentially equally expressive as bounded TAPNs (see [23] for an overview), the translations are exponential and do not allow to perform a direct performance comparison because the modelling capabilities and the treatment of time in TPN and TAPN are very different.

The definition of our extrapolation (abstraction) operator is following [4] where a similar operator was suggested for timed automata; our extension (apart from its adaptation to the TAPN setting) is the handling of dynamic maximum constants depending on the current marking (see also [13] for a dynamic extrapolation on timed automata). The main novelty is our definition of an inclusion operator that incorporates symmetry reduction and works also for nets with monotonicity-breaking features.

2 Timed-Arc Petri Nets

Let $\mathbb{N}$ be the set of natural numbers and let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. By $\mathbb{R}_{\geq 0}$ we denote the set of non-negative real numbers. The set of time intervals $\mathscr{I}$ is given by the abstract syntax ($a \in \mathbb{N}_0, b \in \mathbb{N}$ and $a < b$): $I ::= [a,a] \mid [a,b] \mid [a,b) \mid (a,b] \mid (a,b) \mid [a,\infty) \mid (a,\infty)$. The set of invariant intervals, $\mathscr{I}_{\text{Inv}}$, consists of intervals that include 0.

Let $\mathscr{C} = \{\mathbf{0}, 1, 2, \ldots, n\}$ be a finite set of real-valued clocks whose elements (numbers) represent names of clocks. The clock $\mathbf{0}$ is a special pseudoclock that has always the value 0. A *(clock) valuation* over $\mathscr{C}$ is a function $v : \mathscr{C} \to \mathbb{R}_{\geq 0}$ such that $v(\mathbf{0}) = 0$. The set of all valuations over the clocks $\mathscr{C}$ is denoted by $\mathscr{W}^{\mathscr{C}}$. Let v be a valuation and d a nonnegative real. We let $v+d$ be the valuation such that $(v+d)(i) = v(i)+d$ for every $i \in \mathscr{C} \setminus \{\mathbf{0}\}$ and $(v+d)(\mathbf{0}) = 0$. Further, for a subset of clocks $R \subseteq \mathscr{C}$, we let $v^{R=0}$ be the valuation such that $v^{R=0}(i) = 0$ if $i \in R$ and $v^{R=0}(i) = v(i)$ otherwise.

Let $W \subseteq \mathscr{W}^{\mathscr{C}}$ be a set of valuations and let $R \subseteq \mathscr{C}$. We define the *delay operation* as $W^{\uparrow} = \{v+d \mid v \in W \text{ and } d \in \mathbb{R}_{\geq 0}\}$ and the *reset operation* as $W^{R=0} = \{v^{R=0} \mid v \in W\}$.

A *timed labeled transition system* (TLTS) is a tuple $(S, Lab, \longrightarrow)$ where S is a set of states (or processes), $Lab = Act \cup \mathbb{R}_{\geq 0}$ is a set of labels, consisting of discrete actions and time delays, and $\longrightarrow \subseteq (S \times Lab \times S)$ is the transition relation. We often write $s \xrightarrow{\alpha} s'$ instead of $(s, \alpha, s') \in \longrightarrow$ and if the label is not important, we simply write $s \longrightarrow s'$.

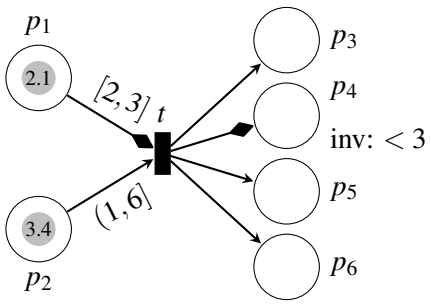


Figure 1: A TAPN with $Pairing(t) = \{(p_1, p_4), (p_2, p_3), (\bot, p_5), (\bot, p_6)\}$

We shall now define the Timed-Arc Petri Net (TAPN) model, restricting ourselves to k-bounded nets (where every reachable marking has at most k tokens). An example of a 4-bounded TAPN is given in Figure 1. It consists of six places (circles), one transition (rectangle) and two tokens of age 2.1 and 3.4 representing the current marking. Input arcs to the transition t contain time intervals and because

both tokens belong to the corresponding interval, the transition can fire, consume the two tokens in p_1 and p_2, and produce a new token of age 0 to each of the places p_3, p_5 and p_6. Because the place p_1 is connected to p_4 via a pair of transport arcs (denoted by a diamond tip), the token of age 2.1 is moved to p_4 while its age is preserved. Should there be more pairs of transport arcs connected to the transition t, we label them with numbers so that the routes on which tokens travel are clearly marked. Finally, note that the place p_4 has an associated age invariant, restricting the possible ages of tokens in the place to strictly less than 3. Should we in the current marking first delay 0.9 time units, both tokens in p_1 and p_2 would still fit into their intervals but the transition t is not enabled any more due to the age invariant in the place p_4.

Definition A TAPN is a 7-tuple $(P, T, IA, OA, c, Type, \iota)$ where

- P is a finite set of places,
- T is a finite set of transitions such that $P \cap T = \emptyset$,
- $IA \subseteq P \times T$ is a finite set of input arcs,
- $OA \subseteq T \times P$ is a finite set of output arcs,
- $c : IA \to \mathscr{I}$ assigns intervals to input arcs,
- $Type : IA \cup OA \longrightarrow \{Normal, Inhib\} \cup \{Transport_i \mid i \in \mathbb{N}\}$ is a function assigning a type to all arcs such that
  - $Type(a) = Inhib \Rightarrow a \in IA \wedge c(a) = [0, \infty)$,
  - $Type(p,t) = Transport_\ell \Rightarrow \exists!(t,p') \in OA \,.\, Type(t,p') = Transport_\ell$ and
  - $Type(t,p') = Transport_\ell \Rightarrow \exists!(p,t) \in IA \,.\, Type(p,t) = Transport_\ell$, and
- $\iota : P \to \mathscr{I}_{inv}$ assigns age invariants to places.

For notational convenience, we write $Type(a) = Transport$ if $Type(a) = Transport_\ell$ for some ℓ. For a transition $t \in T$, we define the *preset* of t as ${}^\bullet t = \{p \in P \mid (p,t) \in IA, Type(p,t) \neq Inhib\}$ and the *postset* of t as $t^\bullet = \{p \in P \mid (t,p) \in OA\}$.

We denote by $P_\perp$ the set $P \cup \{\perp\}$ where $\perp$ is a special symbol representing a pseudo-place that holds the currently unused tokens. The *augmented preset* and *augmented postset* of a transition t are defined as the multisets

$$
\begin{aligned}
{}^\circ t &= \{p_1, \ldots, p_m \mid \{p_1, \ldots, p_\ell\} = {}^\bullet t, p_i = \perp \text{ if } \ell < i \leq m\} \\
t^\circ &= \{p_1, \ldots, p_m \mid \{p_1, \ldots, p_\ell\} = t^\bullet, p_i = \perp \text{ if } \ell < i \leq m\}
\end{aligned}
$$

where $m = \max(|{}^\bullet t|, |t^\bullet|)$. This guarantees that $|{}^\circ t| = |t^\circ|$ for any transition t, a convenient technical detail used in the algorithms. We also extend the definition of c and ι such that $c(\perp, t) = [0, \infty)$ whenever $\perp \in {}^\circ t$ and $\iota(\perp) = [0, \infty)$.

A *token* in a k-bounded TAPN is an element from the set $\{1, 2, \ldots, k\}$. A *marking* is a pair $M = (pl, v)$ where $pl : \{1, 2, \ldots, k\} \to P_\perp$ is the placement function and $v : \{1, 2, \ldots, k\} \to \mathbb{R}_{\geq 0}$ is the age function. The placement determines the current location of each token (it returns $\perp$ if the token is unused) and the age function represents the age of each token. The placement function will be sometimes written as a vector where e.g. $[p_1, p_2, p_1]$ represents the fact that tokens 1 and 3 are located in the place p_1 and token 2 is located in p_2. The set of all markings on a k-bounded TAPN N is denoted by $\mathscr{M}(N)$. A *marked* k-bounded TAPN is a pair $(N, (pl_0, v_0))$ where N is a k-bounded TAPN and (pl_0, v_0) is the initial marking where $v_0(i) = 0$ for all i, $1 \leq i \leq k$.

Since there are always k tokens in any marking (unused ones are in $\bot$), it is for algorithmic purposes convenient to fix for each transition the paths from input to output places. This is formalized in the function $Pairing : T \to 2^{P_\bot \times P_\bot}$ such that for every transition t we have

$$Pairing(t) = \{(p_1,p'_1),\ldots,(p_\ell,p'_\ell) \mid \{p_1,\ldots,p_\ell\} = {}^\circ t, \{p'_1,\ldots,p'_\ell\} = t^\circ \text{ and } Type(p_i,t) = Type(t,p'_j) = Transport_\ell \Rightarrow i = j\}.$$

An example of a possible pairing function is given in Figure 1.

The effect of moving tokens in a placement pl by firing a transition t with the pairing $Pairing(t) = \{(p_1,p'_1),\ (p_2,p'_2),\ldots,(p_\ell,p'_\ell)\}$ is defined in the expected way as follows. Let $IN = \{i_1,i_2,\ldots,i_\ell\} \subseteq \{1,2,\ldots,k\}$ be a set of tokens placed in the places p_1 to p_ℓ and used for firing t. Formally, $pl(i_j) = p_j$ for all $1 \leq j \leq \ell$. The move function $move(pl,IN,t) : \{1,2,\ldots,k\} \to P_\bot$ is now given by

$$move(pl,IN,t)(i) = \begin{cases} pl(i) & \text{if } i \notin IN \\ p'_j & \text{if } i \in IN \text{ such that } i = i_j. \end{cases}$$

Consider Figure 1 and let $pl = \left[p_1,p_2,\bot,\bot\right]$. Then $move(pl,\{1,2,3,4\},t) = \left[p_4,p_3,p_5,p_6\right]$.

Transition Enabledness A transition $t \in T$ is *enabled* by a set of tokens $IN \subseteq \{1,2,\ldots,k\}$ in a marking (pl,v) if

(i) ${}^\circ t = \{pl(i) \mid i \in IN\}$

(ii) $v(i) \in c(pl(i),t)$ for all $i \in IN$

(iii) $Type(pl(i),t) = Transport$ implies $v(i) \in \iota(move(pl,IN,t)(i))$ for all $i \in IN$

(iv) $(pl(i),t) \in IA$ implies $Type(pl(i),t) \neq Inhib$ for all $i \in \{1,2,\ldots,k\} \setminus IN$.

A transition t is hence enabled if there is a token in each of its input places *(i)*, the ages of these tokens fit into the intervals on the input arcs *(ii)*, the age of the token that is moved along a pair of transport arcs does not break the age invariant of the place where is it moved to *(iii)*, and there is no token in any place connected via inhibitor arc to the transition t *(iv)*.

Transition Firing A transition t enabled in a marking (pl,v) by the set of tokens IN can *fire*, producing a marking $(move(pl,IN,t),v^{R=0})$ where $R = \{i \in IN \mid Type(pl(i),t) \neq Transport\}$.

Time Delay A time delay of $d \in \mathbb{R}_{\geq 0}$ time units is possible in a marking (pl,v) if $v(i)+d \in \iota(pl(i))$ for all $i \in \{1,2,\ldots,k\}$. By delaying d time units, we reach the marking $(pl,v+d)$.

The *concrete execution semantics* of a TAPN $N = (P,T,IA,OA,c,Type,\iota)$ is given by a TLTS $T(N) = (\mathscr{M}(N), T \cup \mathbb{R}_{\geq 0}, \longrightarrow)$ where states are markings on N and labels are transition names and time delays. The transition relation $\longrightarrow$ is defined so that $M \xrightarrow{t} M'$ if by firing t in the marking M we reach the marking M', and $M \xrightarrow{d} M'$ if by delaying d time units in the marking M we reach the marking M'.

3 Symbolic Semantics

The concrete execution semantics is not suitable for the actual verification as there are infinitely (in fact uncountably) many reachable markings. Therefore we give a symbolic semantics of k-bounded TAPNs with respect to some given abstraction operator and show that the symbolic semantics preserves the answer to the reachability question.

A *symbolic marking* of a k-bounded TAPN is a pair (pl, W) where $pl : \{1,2,\ldots,k\} \to P_\perp$ is a placement function and $W \subseteq \mathscr{W}^{\mathscr{C}}$ is a set of valuations.

In order to guarantee the finiteness of the state-space in the abstract semantics, we consider abstraction operators that can enlarge (extrapolate) the possible sets of valuations in symbolic markings. Instead of considering global abstraction operators like for example in the timed automata theory (see e.g. [4]), our abstraction operators depend also on the current placement.

Definition An *abstraction operator* is a function $\alpha : [\{1,2,\ldots,k\} \to P_\perp] \times 2^{\mathscr{W}^{\mathscr{C}}} \longrightarrow 2^{\mathscr{W}^{\mathscr{C}}}$ such that $W \subseteq \alpha(pl, W)$ for all symbolic markings (pl, W).

An example of an abstraction operator is the identity abstraction operator α_{id} where $\alpha_{id}(pl, W) = W$ for all symbolic markings (pl, W).

Our aim is of course to find an operator that for a given net abstracts as much as possible. To do so, we use the function $mci : \mathscr{I} \to \mathbb{N}_0$ that returns, for an interval I, the maximum constant different from ∞ appearing in I. Let gc be the maximum constant different from ∞ that appears in intervals or invariants of the given TAPN. The function $mc : P_\perp \to \mathbb{N}_0$ now returns, for each place p, the maximum constant appearing in the guards of outgoing arcs from p or in the invariant of p; if there are transport arcs connected to p, the constant is gc.

$$mc(p) = \begin{cases} gc & \text{if there exists } (p,t) \in IA \text{ s.t. } Type(p,t) = Transport \\ \max\left(mci(\iota(p)), \max\limits_{(p,t)\in IA}(mci(c(p,t)))\right) & \text{otherwise.} \end{cases}$$

Following [4], we proceed to define an equivalence on valuations. The addition in our paper is that we take the placement function into account, thereby allowing for dynamic maximum constants. Let pl be a placement function and let v and v' be valuations. We write $v \equiv_{pl} v'$ if for all $i \in \mathscr{C} \setminus \{\mathbf{0}\}$

1. $v(i) = v'(i)$, or
2. $v(i) > mc(pl(i))$ and $v'(i) > mc(pl(i))$.

Hence two related valuations are indistinguishable from each other in the sense that they can be used to fire the same transitions. Now we can define an abstraction operator based on the relation above.

Definition Let $\alpha_{\equiv}(pl, W) = \{v' \mid v' \equiv_{pl} v \text{ and } v \in W\}$ for a set of valuations $W \subseteq \mathscr{W}^{\mathscr{C}}$ and a placement function pl.

Clearly, $W \subseteq \alpha_{\equiv}(pl, W)$ for any set of valuations $W \subseteq \mathscr{W}^{\mathscr{C}}$ and any placement function pl as the relation is reflexive. For two abstraction operators α and α' we write $\alpha \subseteq \alpha'$ if $\alpha(pl, W) \subseteq \alpha'(pl, W)$ for all placement functions pl and all $W \subseteq \mathscr{W}^{\mathscr{C}}$.

We are now ready to give the symbolic semantics of TAPNs. Let g be a function that takes a placement function pl, a set of tokens IN and a transition t as its arguments (assuming that ${}^\circ t = \{pl(i) \mid i \in IN\}$) and it returns the set of all valuations such that the tokens in IN satisfy all guards on the input arcs of t. Formally, $g(pl, IN, t) = \bigcap_{i \in IN} \{v \in \mathscr{W}^{\mathscr{C}} \mid v(i) \in c(pl(i), t)\}$. Similarly, we define a function I that takes a placement function as its argument and returns the set of all valuations satisfying the age invariants. Formally, $I(pl) = \bigcap_{i \in \mathscr{C}\setminus\{\mathbf{0}\}} \{v \in \mathscr{W}^{\mathscr{C}} \mid v(i) \in \iota(pl(i))\}$.

Symbolic Semantics Let $(N, (pl_0, v_0))$ be a marked k-bounded TAPN and let α be an abstraction operator. The symbolic semantics of $(N, (pl_0, v_0))$ is given by a TLTS $T(N) = (S, L, \rightsquigarrow_\alpha)$ where

- $S = [\{1,2,\ldots,k\} \to P_\perp] \times (2^{\mathscr{W}^{\mathscr{C}}} \setminus \emptyset)$,

- $L = T \cup \{\varepsilon\}$, and
- $(pl, W) \overset{t}{\rightsquigarrow}_\alpha (pl', \alpha(pl', W'))$ if t is a transition and there is a set of tokens IN such that
  - $^\circ t = \{pl(i) \mid i \in IN\}$
  - $pl' = move(pl, IN, t)$
  - $W' \overset{def}{=} (W \cap g(pl, IN, t))^{R=0} \cap I(pl')$ is consistent ($W' \neq \emptyset$) where $R = \{i \in IN \mid Type(pl(i), t) \neq Transport\}$
  - $(pl(i), t) \in IA$ implies $Type(pl(i), t) \neq Inhib$ for all $i \in \{1, \ldots, k\} \setminus IN$
- $(pl, W) \overset{\varepsilon}{\rightsquigarrow}_\alpha (pl, \alpha(pl, W^\uparrow \cap I(pl)))$.

The initial symbolic marking is $(pl_0, \{v_0\})$ where $v_0(i) = 0$ for all $i \in \mathscr{C}$.

Let us define $\overset{T}{\rightsquigarrow}_\alpha \overset{def}{=} \cup_{t \in T} \overset{t}{\rightsquigarrow}_\alpha$. We can now state the main theorem of this section, which establishes soundness and completeness of the symbolic semantics for any abstraction operator between α_{id} and $\alpha_\equiv$. In fact, we allow to dynamically change the abstraction operators during a computation in the symbolic semantics. Hence we consider a new transition relation $\rightsquigarrow_{\alpha_{id}, \alpha_\equiv} \overset{def}{=} \bigcup_{\alpha_{id} \subseteq \alpha \subseteq \alpha_\equiv} \rightsquigarrow_\alpha$ allowing us to apply in any step an arbitrary abstraction operator between the identity and $\alpha_\equiv$.

Theorem 3.1 *Let $(N, (pl_0, v_0))$ be a marked k-bounded TAPN. Then*

- (Soundness) $(pl_0, \{v_0\}) \rightsquigarrow^*_{\alpha_{id}, \alpha_\equiv} (pl, W)$ *implies that there exists a valuation $v \in W$ such that $(pl_0, v_0) \longrightarrow^* (pl, v)$, and*
- (Completeness) $(pl_0, v_0) \longrightarrow^* (pl, v)$ *implies, for any abstraction operator α where $\alpha_{id} \subseteq \alpha \subseteq \alpha_\equiv$, that $(pl_0, \{v_0\}) \overset{\varepsilon}{\rightsquigarrow}_\alpha \circ (\overset{T}{\rightsquigarrow}_{\alpha_{id}} \circ \overset{\varepsilon}{\rightsquigarrow}_\alpha)^* (pl, W)$ for some W where $v \in W$.*

Note that the completeness part of the theorem imposes that the symbolic semantics can reach the given placement via a strictly alternating sequence of time elapsing and transition firing steps where the transition firing steps are not extrapolated (using the identity abstraction operator); this reflects how the successors are computed in the reachability algorithm discussed in Section 6.

4 Extrapolation via DBMs

For the use in our reachability algorithm, we need to represent infinite sets of valuations W in a finite way. However, it is not known how to effectively deal directly with the $\alpha_\equiv$ abstraction operator. Instead, we suggest a slightly less general abstraction (extrapolation) operator and a way to finitely represent infinite sets of valuations in order to guarantee a finite and effectively searchable state-space of symbolic markings.

For this purpose we use Difference Bound Matrices (DBM), a well-known technique for verification of real-time systems (see e.g. [5, 10]) that allows us to store constraints on single clocks and on differences of two clocks in a compact matrix-based data structure.

Difference Bound Matrix (DBM) A *Difference Bound Matrix D* over the set of clocks $\mathscr{C}$ is a $|\mathscr{C}| \times |\mathscr{C}|$ matrix such that

$$D_{ij} \in (\mathbb{Z} \times \{<, \leq\}) \cup \{(\infty, <)\}$$

where $i, j \in \mathscr{C}$ and for all $i \in \mathscr{C}$ we have

1. if $D_{\mathbf{0}i} = (m, \lhd)$ then $m \leq 0$, and $\lhd \in \{<, \leq\}$,

2. if $D_{i\mathbf{0}} = (m, \lhd)$ then $m \geq 0$, and $\lhd \in \{<, \leq\}$, and
3. $D_{ii} = (0, \leq)$.

A *solution* to a DBM D is a valuation v such that for all $i, j \in \mathscr{C}$ we have $v(i) - v(j) \lhd m$ where $D_{ij} = (m, \lhd)$. The set of all solutions to a DBM D (alternatively, the zone over D) is denoted by $[D]$.

We refer to the elements D_{ij} as *bounds*. A bound $D_{\mathbf{0}i} = (m, \lhd)$ where $m \leq 0$ (by Condition 1) is called the *lower bound* for the clock i. Such a constraint means $v(\mathbf{0}) - v(i) \lhd m$ for any valuation $v \in [D]$, which is equivalent to $-m \lhd v(i)$. Similarly, a bound $D_{i\mathbf{0}} = (m, \lhd)$ where $m \geq 0$ (by Condition 2) is called the *upper bound* for the clock i and it means that $v(i) - v(\mathbf{0}) \lhd m$ which is the same as $v(i) \lhd m$. Finally, a bound D_{ij} where $i \neq \mathbf{0} \neq j$ is called a *diagonal constraint*.

For notational convenience, we introduce an alternative notation *lb* and *ub* for the lower and upper bound of a clock i. Formally, $lb_D(i) = (-m, \lhd)$ if $D_{\mathbf{0}i} = (m, \lhd)$ and $ub_D(i) = D_{i\mathbf{0}}$. We further define a notation for the individual elements in a bound such that $lb_D^{\eta}(i) = m$ and $lb_D^{\lhd}(i) = \lhd$ if $lb_D(i) = (m, \lhd)$. We use the same notation ub_D^{η} and $ub_D^{\lhd}$ also for upper bounds.

A DBM D is *consistent* if $[D] \neq \emptyset$. We say that D is in *canonical form* if $D_{ij} \preceq D_{ik} + D_{kj}$ for all $i, j, k \in \mathscr{C}$. It is well known that for every consistent DBM D there is a unique canonical DBM D^c such that $[D] = [D^c]$ [10].

We now define a variant of one of the abstraction (extrapolation) operators on DBMs in order to abstract sets of valuations represented by a DBM. The definition is inspired by [4], the main difference being the use of dynamic maximum constants in our operator.

Extrapolation The extrapolation of a canonical DBM D in a placement pl is the DBM D', called $ext_{pl}(D)$, and defined as follows (here $i, j \in \mathscr{C} \setminus \{\mathbf{0}\}$ such that $i \neq j$):

1. $D' := D$
2. if $mc(pl(i)) < lb_D^{\eta}(i)$ then $lb_{D'}(i) := (mc(pl(i)), <)$ and $ub_{D'}(i) := (\infty, <)$
3. if $ub_D^{\eta}(i) > mc(pl(i))$ then $ub_{D'}(i) := (\infty, <)$
4. if $mc(pl(i)) < lb_D^{\eta}(i)$ or $mc(pl(j)) < lb_D^{\eta}(j)$ then $D'_{ij} := (\infty, <)$
5. if $D_{ij} = (m, \lhd)$ and $m > mc(pl(i))$ then $D'_{ij} := (\infty, <)$

Intuitively, the extrapolation works by removing all upper bounds greater than the maximum constant of a given place and by replacing any lower bound greater than the maximum constant with the value $(mc(pl(i)), <)$. Additionally, whenever the lower bound is above the maximum constant of a given place, any diagonal constraint involving that clock are also removed. An example of a DBM D and its extrapolation $ext_{pl}(D)$ together with their graphical representations (clock 1 is on the x-axis and clock 2 on the y-axis) is given in Figure 2. We can see that the extrapolation operator enlarges the set of valuations represented by D such that there are only finitely many extrapolated DBMs.

Lemma 4.1 *The set $\{ext_{pl}(D) \mid D$ is a canonical DBM$\}$ is finite.*

We can now conclude with the main result stating that the extrapolation provides an abstraction which is between identity and $\alpha_{\equiv}$; a crucial and nontrivial fact needed for proving correctness of the reachability algorithm.

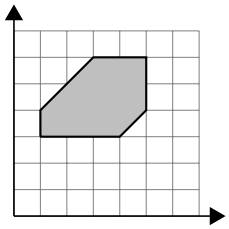
Theorem 4.2 *Let D be a canonical DBM and let pl be a placement function. Then $[D] \subseteq [ext_{pl}(D)] \subseteq \alpha_{\equiv}(pl, [D])$.*

$(0,\leq)$	$(-1,\leq)$	$(-3,\leq)$
$(5,\leq)$	$(0,\leq)$	$(1,\leq)$
$(6,\leq)$	$(3,\leq)$	$(0,\leq)$

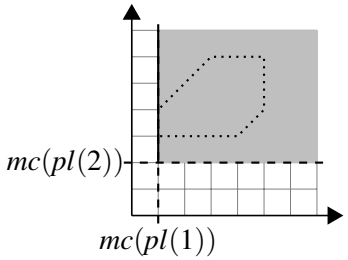
(a) Canonical DBM D

$(0,\leq)$	$(-1,\leq)$	$(-2,<)$
$(\infty,<)$	$(0,\leq)$	$(\infty,<)$
$(\infty,<)$	$(\infty,<)$	$(0,\leq)$

(b) The DBM $ext_{pl}(D)$



(c) The zone $[D]$



(d) The zone $[ext_{pl}(D)]$

Figure 2: Example of the extrapolation operator for $mc(pl(1)) = 1, mc(pl(2)) = 2$

5 Monotonicity of Bounded TAPNs

It is a well-known fact that the behaviour of the basic TAPN model is monotonic [11] with respect to the standard marking inclusion, intuitively meaning that adding more tokens to the net does not restrict its behaviour. However, the use of age invariants and inhibitor arcs breaks the monotonicity property [14]. In this section, we introduce a more refined inclusion relation on symbolic markings that preserves monotonicity even in the presence of age invariants and inhibitor arcs. Moreover, the inclusion relation allows for reordering of tokens in the net and hence it implements the symmetry reduction. The inclusion relation is then exploited in the reachability algorithm presented in Section 6.

Let us fix a marked k-bounded TAPN $(N,(pl_0,v_0))$. For a place $p \in P$, we define a boolean predicate $untimed(p) \equiv (\iota(p) = [0,\infty)) \wedge \forall t \in p^\bullet . (Type(p,t) \neq Transport \wedge c(p,t) = [0,\infty))$. If the predicate is true, we do not need to keep track of the ages of tokens in this place. For a symbolic marking M we now define the set INC_M representing the set of tokens eligible for the inclusion checking.

Definition Let $M = (pl,W)$ be a symbolic marking. We define $INC_M \subseteq \{1,2,\ldots,k\}$ as the largest subset of tokens such that for any token $i \in INC_M$,

1. $pl(i) \neq \bot$,
2. $\iota(pl(i)) = [0,\infty)$,
3. $pl(i)$ has no outgoing inhibitor arcs, and
4. either $untimed(pl(i))$ or
   - $\inf(V_i) \in V_i \Rightarrow mc(pl(i)) < \inf(V_i)$, or
   - $\inf(V_i) \notin V_i \Rightarrow mc(pl(i)) \leq \inf(V_i)$,

   where $V_i = \{v(i) \mid v \in W\}$.

Let us briefly comment on Condition 4. If a place is untimed then the ages of tokens in that place are irrelevant and we can consider them for inclusion checking. Otherwise, the lower bound of clock i in

W is calculated by $\inf(V_i)$ and the two subconditions distinguish whether this bound is included or not[1]. The point is that if the lower-bound for the token i is above the maximum constant for the place where i is placed, then its concrete age is irrelevant for the firing of transitions.

Let $P_{inc} \subseteq P$ be a set of places that we want to consider for the inclusion checking (typically we set $P_{inc} = P$ but the user can restrict some places from the application of the inclusion operator by excluding them from P_{inc}). We can now partition all tokens in the marking $M = (pl, W)$ into three categories

- $inc(M) = INC_M \cap \{i \mid pl(i) \in P_{inc}\}$
- $bot(M) = \{i \mid pl(i) = \bot\}$
- $eq(M) = \{1,2,\ldots,k\} \setminus (bot(M) \cup inc(M))$

where $inc(M)$ contains all tokens eligible for inclusion checking, $bot(M)$ contains all unused tokens and $eq(M)$ is the set of all tokens that have to be checked for equality. Let us now introduce some notation. Let pl be a placement function, p a place and let $X \subseteq \{1,2,\ldots,k\}$ be a set of tokens. We define $count^X_{pl}(p) = |\{i \in X \mid pl(i) = p\}|$. Intuitively, $count^X_{pl}(p)$ tells us how many tokens from X are in the place p. We are now ready to introduce the refined ordering relation.

Inclusion Ordering Let $M = (pl, W)$ and $M' = (pl', W')$ be symbolic markings. We say that M is included in M', written $M \sqsubseteq M'$, if

1. There exists a bijection $h : eq(M) \to eq(M')$ such that
   (a) $pl(i) = pl'(h(i))$ for all $i \in eq(M)$,
   (b) for all $v \in W$ there exists a $v' \in W'$ such that for all $i \in eq(M)$
      (i) $v(i) = v'(h(i))$, or
      (ii) $v(i) > mc(pl(i))$ and $v'(h(i)) > mc(pl'(h(i)))$,
2. $count^{inc(M)}_{pl}(p) \leq count^{inc(M')}_{pl'}(p)$ for all $p \in P$.

Hence two symbolic markings M and M' are related by $\sqsubseteq$, if they agree on the sets $eq(M)$ and $eq(M')$ via the bijection h (this gives us the possibility to employ symmetry reduction), and moreover, the number of tokens in any place p from the set $inc(M')$ in the marking M' must be larger than or equal to the number of tokens in the place p in the marking M. We finish this section by a theorem proving monotonicity with respect to the ordering relation $\sqsubseteq$ for any abstraction operator below $\alpha_{\equiv}$.

Theorem 5.1 *Let α be an abstraction operator such that $\alpha_{id} \subseteq \alpha \subseteq \alpha_{\equiv}$ and $M_1, M_2 \in \mathscr{M}_\alpha(N, (pl_0, \{v_0\}))$ be reachable symbolic markings such that $M_1 \sqsubseteq M_2$. If $M_1 \rightsquigarrow_\alpha M_1'$ then $M_2 \rightsquigarrow_\alpha M_2'$ for some M_2' such that $M_1' \sqsubseteq M_2'$.*

6 Implementation of the Reachability Algorithm

Before we present the reachability algorithm, let us first introduce a reachability fragment of CTL that is used in the algorithm. A formula of the logic is given by the abstract syntax:

$$\begin{aligned} \phi &::= \mathsf{EF}\,\psi \mid \mathsf{AG}\,\psi \\ \psi, \psi_1, \psi_2 &::= (p \bowtie n) \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \end{aligned} \tag{1}$$

where $p \in P$, $n \in \mathbb{N}_0$ and $\bowtie \in \{<, \leq, =, \neq, \geq, >\}$.

[1] In the DBM representation we can read these bounds directly from the matrix.

The semantics of formulae is given in terms of a TLTS $(S, Lab, \longrightarrow)$ and a labeling function $\mu : S \to 2^{AP}$ assigning sets of true atomic propositions to states. We define the set of atomic propositions AP and the labeling function μ as $AP \stackrel{def}{=} \{(p \bowtie n) \mid p \in P, n \in \mathbb{N}_0 \text{ and } \bowtie \in \{<, \leq, =, \neq, \geq, >\}\}$ and $\mu(M) \stackrel{def}{=} \{(p \bowtie n) \mid count_{pl}^{\{1,2,\ldots,k\}}(p) \bowtie n \text{ and } \bowtie \in \{<, \leq, =, \neq, \geq, >\}\}$. The intuition is that a proposition $(p \bowtie n)$ is true in a marking M if the number of tokens in the place p satisfies the proposition with respect to n. Since atomic propositions only depend on the discrete part of a marking, we adopt the same definition of μ for symbolic markings. For a state $s \in S$ and a formula ϕ, we define the satisfaction relation $s \models \phi$ inductively as follows:

$$
\begin{aligned}
s &\models (p \bowtie n) \text{ iff } (p \bowtie n) \in \mu(s) \\
s &\models \neg\psi \quad \text{iff } s \not\models \psi \\
s &\models \psi_1 \wedge \psi_2 \text{ iff } s \models \psi_1 \text{ and } s \models \psi_2 \\
s &\models \psi_1 \vee \psi_2 \text{ iff } s \models \psi_1 \text{ or } s \models \psi_2 \\
s &\models \mathsf{EF}\,\psi \quad \text{iff } s \longrightarrow^* s' \text{ and } s' \models \psi \\
s &\models \mathsf{AG}\,\psi \quad \text{iff } s \not\models \mathsf{EF}\,\neg\psi \quad .
\end{aligned}
$$

As the AG and EF temporal operators are dual, it is enough to design an algorithm for deciding the validity of $\mathsf{EF}\,\psi$. Note that because the predicates do not allow us to query the ages of tokens in the net, the presence of age invariants in the TAPN model adds an expressive power (otherwise we could conjunct the age invariants with the intervals on input arcs and add to the formulae the requirement that no place contains any token exceeding the invariant bound).

We say that a place p in a boolean predicate ψ defined according to Equation (1) is *monotonicity-breaking* if ψ contains an atomic proposition of the form $p < n$, $p \leq n$, $p = n$ or $p \neq n$. In other words, the inequality imposes some upper bound or an exact comparison to a concrete number in the place p.

Lemma 6.1 *Let M and M' be symbolic markings and let ψ be a boolean predicate defined by Equation* (1) *and let the set P_{inc} of inclusion places do not contain any monotonicity-breaking place. If $M \models \psi$ and $M \sqsubseteq M'$ then $M' \models \psi$.*

Proof By structural induction on ψ. The induction step is trivial; we discuss here only the base case for a proposition of the formi $\psi = p \bowtie n$. Let (pl_1, W_1) and (pl_2, W_2) be symbolic markings such that $(pl_1, W_1) \sqsubseteq (pl_2, W_2)$. Let $(pl_1, W_1) \models \psi$.

If p is a monotonicity-breaking place than $p \notin P_{inc}$ and all tokens in the place p belong to the set $eq((pl_1, W_1))$. By Condition 1 of the inclusion ordering there exists a bijection h such that for all $i \in eq((pl_1, W_1))$ we have $pl_1(i) = pl_2(h(i))$ and hence in the marking (pl_2, W_2) the number of tokens in the place p is equal to the number of tokens in the place p in the marking (pl_1, W_1) and we get $(pl_2, W_2) \models \psi$.

If p is not a monotonicity-breaking place, the constraint on p has the form $p \geq n$ or $p > n$. If the tokens in the place p belong to $eq((pl_1, W_1))$ we are done by the arguments as above. If the tokens in the place p belong to $inc((pl_1, W_1))$ then by Condition 2 of the inclusion ordering the number of tokens placed in p in the marking (pl_2, W_2) is at least the number of tokens in the marking (pl_1, W_1) and because the proposition on p states only a lower-bound, we can again conclude that $(pl_2, W_2) \models \psi$. ∎

In order to present an efficient reachability algorithm, we need a finite representation for the potentially infinite sets of valuations discussed in Section 5. We will thus use DBMs. However, we have to implement the operations used on the sets of valuations, such as delay, clock reset and intersection, on

DBMs. Similarly, we need to define a DBM which represents a guard or invariant of the form $i \in [\![a,b]\!]$ where i is a clock and $[\![a,b]\!]$ is a well-formed interval—here $[\![$ is either closed or open left parenthesis and similarly for $]\!]$.

Proposition 6.2 *Let D_1 and D_2 be canonical DBMs over the clocks $\mathscr{C}$. Then the following operations and DBMs can be computed efficiently*

1. *(Delay) $D_1^{\uparrow}$ is a canonical DBM s.t. $[D_1^{\uparrow}] = [D_1]^{\uparrow}$.*
2. *(Reset) $D_1^{R=0}$ where $R \subseteq \mathscr{C}$ is a canonical DBM s.t. $[D_1^{R=0}] = [D_1]^{R=0}$.*
3. *(Intersection) $D_1 \cap D_2$ is a canonical DBM s.t. $[D_1 \cap D_2] = [D_1] \cap [D_2]$.*
4. *(Interval DBM) $D_{i \in [\![a,b]\!]}$ is a canonical DBM s.t. $[D_{i \in [\![a,b]\!]}] = \{v \in \mathscr{W}^{\mathscr{C}} \mid v(i) \in [\![a,b]\!]\}$.*
5. *(Discrete Inclusion) Let pl_1 and pl_2 be placement functions. The expression $(pl_1, [D_1]) \sqsubseteq (pl_2, [D_2])$ can be computed efficiently.*

All these operations can be efficiently implemented for DBMs (see e.g. [5, 20]) for details on operations 1–4; the fifth operator can be implemented using DBMs, as showed in the full version of the paper.

We can now perform a standard search through the state-space of symbolic markings using the passed/waiting list approach. We start by adding the initial marking to the waiting list. As long as the waiting list is nonempty, a symbolic marking M is removed from the waiting list, added to the passed list, and all symbolic extrapolated successors of M are explored. If a successor M' of M is below (w.r.t. the ordering $\sqsubseteq$) some marking on the passed or waiting list, we ignore it. Otherwise we add M' to the waiting list and remove from the waiting and passed lists all markings that are below M'. We stop with a positive answer once we find a marking satisfying a property we are searching for. If the whole state-space is searched without finding such a marking, we return a negative answer. The search is performed only upto k tokens in the net where this number is supplied by the user (it is undecidable whether there is some k such that the net is k-bounded [14]). If the net is k-bounded for the given k (this can be automatically verified) then this gives a conclusive answer, otherwise the search can give a conclusive answer only if it finds a marking satisfying the given property.

The successor generation algorithm is presented in Algorithm 1 and the reachability algorithm is given in Algorithm 2. Observe, as remarked above, that the algorithm will discard any generated successor marking if a larger marking is already present in the PASSED or WAITING list (line 11). Similarly, if a generated successor marking is larger than some marking in the PASSED or WAITING list, then it will remove all such smaller markings from the PASSED and WAITING list (lines 12 to 13).

Lemma 6.3 *Algorithm 2 terminates.*

Proof Let N be a k-bounded TAPN. We must argue that the state-space of the symbolic semantics is finite. Since N is a k-bounded TAPN, it follows that there are only finitely many placement functions. Further, from Theorem 4.1 we know that there are only finitely many extrapolated DBMs for a given placement function. Thus, we may conclude that there are only finitely many symbolic markings in the symbolic semantics using the extrapolation operator. Observe that Algorithm 2 will add each symbolic marking to the *WAITING* list at most once. Thus, it follows that the algorithm terminates. ∎

Lemma 6.4 *If Algorithm 2 returns "YES", then $(pl_0, v_0) \models \mathsf{EF}\,\psi$.*

Proof Assume that Algorithm 2 returns "YES". We must show that $(pl_0, v_0) \longrightarrow^* (pl, v)$ such that $(pl, v) \models \psi$. We define $\alpha_{ext}(pl, [D]) \stackrel{\text{def}}{=} [ext_{pl}(D)]$ for any placement function pl and canonical DBM D (for any set of valuations W that cannot be represented by a DBM we assume $\alpha_{ext}(pl, W) \stackrel{\text{def}}{=} W$).

Algorithm 1: Successor generation algorithm

Name: **succ**$(N,(pl,D))$;
Input: A k-bounded TAPN N and a symbolic marking (pl,D).
Output: The set of successor markings for (pl,D).
begin
  successors := $\emptyset$;
  forall the $t \in T$ **do**
    Let $\Delta := \{i \in \{1,2,\ldots,k\} \mid pl(i) \in {}^\circ t\}$;
    forall the *sets* $IN \subseteq \Delta$ *where* ${}^\circ t = \{pl(i) \mid i \in IN\}$ *and* $\forall i \in \{1,2,\ldots,k\} \setminus IN\,.\,(pl(i),t) \in IA \Rightarrow Type(pl(i),t) \neq Inhib$ **do**
      Let $R := \{i \in IN \mid Type(pl(i),t) \neq Transport\}$;
      pl' := move(pl, IN, t);
      $D' := \left(D \cap \bigcap_{i \in IN} D_{i \in c(pl(i),t)}\right)^{R=0} \cap \bigcap_{i \in \{1,\ldots,k\}} D_{i \in \iota(pl'(i))}$;
      if D' *is consistent* **then**
        successors := successors $\cup \left\{\left(pl', ext_{pl'}\left((D')^{\uparrow} \cap \bigcap_{i \in \{1,2,\ldots,k\}} D_{i \in \iota(pl'(i))}\right)^{c}\right)\right\}$;
  return successors;

Algorithm 2: Reachability algorithm

Name: **Reach**$(N,(pl_0,v_0),\mathsf{EF}\,\psi)$;
Input: A marked k-bounded TAPN $(N,(pl_0,v_0))$, a formula $\mathsf{EF}\,\psi$ and a set $P_{inc} \subseteq P$ not containing any monotonicity-breaking place in ψ.
Output: YES if $(pl_0,v_0) \models \mathsf{EF}\,\psi$, NO otherwise.
begin
  $PASSED := \emptyset$;
  Create DBM D_0 such that $[D_0] = \{v_0\}$;
  if $(pl_0, ext_{pl_0}\left(D_0^{\uparrow} \cap \bigcap_{i \in \{1,2,\ldots,k\}} D_{i \in \iota(pl_0(i))}\right)^{c}) \models \psi$ **then return** YES;
  $WAITING := \{(pl_0, ext_{pl_0}\left(D_0^{\uparrow} \cap \bigcap_{i \in \{1,2,\ldots,k\}} D_{i \in \iota(pl_0(i))}\right)^{c})\}$;
  while $WAITING \neq \emptyset$ **do**
    Remove some (pl,D) from $WAITING$;
    $PASSED := PASSED \cup \{(pl,D)\}$;
    forall the $(pl',D') \in succ(N,(pl,D))$ **do**
      if $\neg\exists (pl'',D'') \in PASSED \cup WAITING\,.\,(pl',[D']) \sqsubseteq (pl'',[D''])$ **then**
        $PASSED := PASSED \setminus \{(pl'',D'') \in PASSED \mid (pl'',[D'']) \sqsubseteq (pl',[D'])\}$;
        $WAITING := WAITING \setminus \{(pl'',D'') \in WAITING \mid (pl'',[D'']) \sqsubseteq (pl',[D'])\}$;
        if $(pl',D') \models \psi$ **then return** YES;
        $WAITING := WAITING \cup \{(pl',D')\}$;
  return NO;

Delay	TAPAAL	TAPAAL incl.	Broadcast	Deg.2 Broadcast
28	10.8	10.4	11.6	11.6
24	12.1	12.0	102.3	48.8
20	17.0	16.4	456.2	88.0
16	92.6	90.7	207.4	137.7

Table 1: PMS case study scaled by the sampling delay (time in seconds)

Since the algorithm returned "YES", it must have found some symbolic marking (pl,D) such that $(pl,D) \models \psi$. Observe that Algorithm 2 (and Algorithm 1) will alternate between using the identity abstraction for discrete transition firings and α_{ext} for time delays. Thus, from the way the algorithm searches through the state-space, we may conclude that there must exist symbolic markings $(pl_1,D_1),(pl_2,D_2),\ldots,(pl,D)$ such that

$$(pl_0,[D_0]) \rightsquigarrow_{\alpha_{id},\alpha_{ext}} (pl_1,[D_1]) \rightsquigarrow_{\alpha_{id},\alpha_{ext}} \cdots \rightsquigarrow_{\alpha_{id},\alpha_{ext}} (pl,[D])$$

where $[D_0] = \{v_0\}$. By Theorem 4.2 and Theorem 3.1, we have that $\rightsquigarrow_{\alpha_{ext}}$ is sound. Thus, there exists a concrete marking (pl,v) such that $(pl_0,v_0) \longrightarrow^* (pl,v)$ and $v \in [D]$. Since atomic propositions depend only on the discrete part of a marking (placement function), it follows that $(pl,v) \models \psi$. ∎

Lemma 6.5 *If $(pl_0,v_0) \models \mathsf{EF}\,\psi$ then Algorithm 2 returns "YES".*

Proof Assume that $(pl_0,v_0) \models \mathsf{EF}\,\psi$. This means that $(pl_0,v_0) \longrightarrow^* (pl,v)$ and $(pl,v) \models \psi$. We must show that Algorithm 2 returns "YES". We define $\alpha_{ext}(pl,[D]) \stackrel{\text{def}}{=} [ext_{pl}(D)]$ as before. By Theorem 4.2 and Theorem 3.1, we get that $\rightsquigarrow_{\alpha_{ext}}$ is complete. Thus, there exists a symbolic marking $(pl,[D]) \models \psi$ such that $(pl_0,[D_0]) \stackrel{\varepsilon}{\rightsquigarrow}_{\alpha_{ext}} \circ (\stackrel{T}{\rightsquigarrow}_{\alpha_{id}} \circ \stackrel{\varepsilon}{\rightsquigarrow}_{\alpha_{ext}})^* (pl,[D])$ where $[D_0] = \{v_0\}$ and $v \in [D]$.

We will now argue that Algorithm 2 will find a symbolic marking (pl',D') such that $(pl,[D]) \sqsubseteq (pl',[D'])$. It is easy to see that Algorithm 2 together with Algorithm 1 implements a symbolic exploration of the form $\stackrel{\varepsilon}{\rightsquigarrow}_{\alpha_{ext}} \circ (\stackrel{T}{\rightsquigarrow}_{\alpha_{id}} \circ \stackrel{\varepsilon}{\rightsquigarrow}_{\alpha_{ext}})^*$. However, notice that the algorithm discards some of the discovered symbolic markings (lines 11 to 13 in Algorithm 2). If the algorithm finds a symbolic marking (pl',D') for which $(pl',[D']) \sqsubseteq (pl'',[D''])$ for some (pl'',D'') in the PASSED or WAITING list, it will discard (pl',D') (line 11). Similarly, if $(pl'',[D'']) \sqsubseteq (pl',[D'])$ for some (pl'',D'') in the PASSED or WAITING list, it will remove all markings $(pl'',[D'']) \sqsubseteq (pl',[D'])$ from both the PASSED and WAITING list (lines 12 to 13). However, by Theorem 5.1 it is safe to skip these symbolic markings since the future behaviour of the smaller symbolic markings is included in the larger symbolic marking. Thus, it follows that Algorithm 2 will find a symbolic marking (pl',D') such that $(pl,[D]) \sqsubseteq (pl',[D'])$. By Theorem 6.1, we have that if the smaller marking satisfies ψ then the larger marking $(pl',[D'])$ also satisfies ψ. Thus, Algorithm 2 returns "YES". ∎

The correctness of the reachability algorithm is hence established.

7 Experiments

We implemented the reachability algorithm in `C++` and fully integrated it into the tool TAPAAL [9] (www.tapaal.net), an open-source and platform-independent editor, simulator and verifier of extended timed-arc Petri nets. In order to document the performance of our proposed algorithm, we present two larger case studies of Patient Monitoring System (PMS) and a communication protocol from the WS-Business Activity standard [19]. Both models can be downloaded from the tool's homepage.

Messages	TAPAAL	TAPAAL incl.	Broadcast	Deg.2 Broadcast
2	2.5	0.6	2.9	2.3
3	11.6	2.1	12.0	7.8
4	46.3	8.0	46.2	24.9
5	164.1	29.1	165.0	73.5
6	>400.0	109.6	>400.0	197.7
7	>400.0	330.4	>400.0	>400.0

Table 2: BAwPC scaled by number of retransmission messages (time in seconds)

The patient monitoring system (PMS) is a case study taken from [8]. The system monitors the pulse rate and the level of oxygen saturation via sensors applied on the skin of a patient. It consists of three components: sampling subsystem, signal analyzer and alarm. The purpose of the PMS model is to verify that abnormal situations dangerous for the patient's health are detected within given deadlines. We have verified the model for deadline violation both in the sampling component and the signal analyzer. The sampling delay has been varied from 28 down to 16 seconds. This increased the complexity of the verification, as the queries were still satisfied and the whole state-space had to be searched.

In the second case study we verify the correctness of one of the web services coordination protocols called Business Activity with Participant Completion (BAwPC) [19]. Our model is based on the work presented in [17] where an enhanced protocol that avoids reaching any invalid states is given. We modelled the protocol in TAPAAL and considered asynchronous communication where messages can be lost; the model is scaled by the number of extra messages that can be used for retransmissions. The protocol is correct and hence the whole state-space is searched.

We compare the performance of our implementation with the UPPAAL engine where the timed automata models were obtained by automatic translations (called broadcast and degree 2 broadcast; see [15, 16] for the details) from the TAPN models. We remark that the verification times of the translated TAPN models are in general comparable with native UPPAAL models and in some examples the translated models verify even faster than the native ones [16]. A direct comparison with other Petri net tools extended with time like Romeo [12] and TINA [6] is not possible due to the radically different semantics of the Petri net models used in these tools.

All experiments were run on Macbook Pro with 2.7 GHz Intel Core i7 with 8 GB RAM using BFS search strategy and the results are presented in Tables 1 and 2. The column TAPAAL refers to our algorithm where the set of inclusion places has been set to empty and TAPAAL incl. is our algorithm with the largest possible inclusion set. The user has the possibility to choose between these algorithms (or even manually select the concrete inclusion places) because for example in the case of 1-safe Petri nets where the inclusion is only rarely applied, the algorithm with the maximum inclusion can be slower due to the implementation overhead connected with inclusion checking of markings on the passed and waiting list. Indeed, in situations like in Table 1 the full inclusion checking is not that beneficial opposite to nets like in Table 2 where we have many tokens (messages) in the same place.

8 Conclusion

We presented a reachability algorithm for extended timed-arc Petri nets and implemented it within the tool TAPAAL. The algorithm includes efficient extrapolation and symmetry reduction techniques that show a very encouraging performance even on larger case-studies. We would like to emphasize the fact that all features that are implemented in the tool are formally defined and proved correct. We believe that

our tool, available at www.tapaal.net, is one of a rather few reasonably-sized model checkers with a complete correctness proof taking into account all implemented optimizations and reduction techniques. In the future work we shall look at extending the technique to liveness properties and at further performance improvements by using for example the LU-extrapolation [4].

Acknowledgements. We would like to thank the anonymous reviewers for their comments.

Bibliography

[1] P.A. Abdulla, J. Deneux, P. Mahata & A. Nylén (2007): *Using Forward Reachability Analysis for Verification of Timed Petri Nets*. Nordic J. of Computing 14, pp. 1–42.

[2] P.A. Abdulla & A. Nylén (2001): *Timed Petri Nets and BQOs*. In: Proc. of ICATPN'01, LNCS 2075, Springer, pp. 53–70, doi:10.1007/3-540-45740-2_5.

[3] Rajeev Alur & David L. Dill (1994): *A theory of timed automata*. Theor. Comput. Sci. 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[4] G. Behrmann, P. Bouyer, K. G. Larsen & R. Pelánek (2006): *Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata*. STTT 8(3), pp. 204–215, doi:10.1007/s10009-005-0190-0.

[5] J. Bengtsson & W. Yi (2003): *Timed Automata: Semantics, Algorithms and Tools*. In: Lectures on Concurrency and Petri Nets, ACPN'03, LNCS 3098, Springer, pp. 87–124, doi:10.1007/978-3-540-27755-2_3.

[6] B. Berthomieu & F. Vernadat (2006): *Time Petri Nets Analysis with TINA*. In: Proc. of (QEST'06), IEEE Computer Society, Washington, DC, USA, pp. 123–124, doi:10.1109/QEST.2006.56.

[7] T. Bolognesi, F. Lucidi & S. Trigila (1990): *From Timed Petri Nets to Timed LOTOS*. In: Proc. of PSTV'90, North-Holland, pp. 395–408.

[8] F. Cicirelli, A. Furfaro & L. Nigro (2012): *Model checking time-dependent system specifications using Time Stream Petri Nets and UPPAAL*. Applied Mathematics and Computation 218(16), pp. 8160 – 8186, doi:10.1016/j.amc.2012.02.018.

[9] A. David, L. Jacobsen, M. Jacobsen, K.Y. Jørgensen, M.H. Møller & J. Srba (2012): *TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets*. In: Proc. of (TACAS'12), LNCS 7214, Springer-Verlag, pp. 492–497, doi:10.1007/978-3-642-28756-5_36.

[10] D. L. Dill (1989): *Timing Assumptions and Verification of Finite-State Concurrent Systems*. In: Automatic Verification Methods for Finite State Systems, LNCS 407, Springer, pp. 197–212, doi:10.1007/3-540-52148-8_17.

[11] A. Finkel & P. Schnoebelen (2001): *Well-Structured Transition Systems Everywhere!* Theoretical Computer Science 256(1-2), pp. 63–92, doi:10.1016/S0304-3975(00)00102-X.

[12] G. Gardey, D. Lime, M. Magnin & O. Roux (2005): *Romeo: A Tool for Analyzing Time Petri Nets*. In: Proc. of CAV'05, LNCS 3576, Springer-Verlag, pp. 261–272, doi:10.1007/11513988_41.

[13] F. Herbreteau, D. Kini, B. Srivathsan & I. Walukiewicz (2011): *Using non-convex approximations for efficient analysis of timed automata*. In: Proc. of FSTTCS'11, LIPIcs 13, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 78–89, doi:10.4230/LIPIcs.FSTTCS.2011.78.

[14] L. Jacobsen, M. Jacobsen & M. H. Møller (2009): *Undecidability of Coverability and Boundedness for Timed-Arc Petri Nets with Invariants*. In: *Proc. of MEMICS'09*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 1–8, doi:10.4230/DROPS.MEMICS.2009.2346.

[15] L. Jacobsen, M. Jacobsen, M.H. Møller & J. Srba (2010): *A Framework for Relating Timed Transition Systems and Preserving TCTL Model Checking*. In: *Proc. of EPEW'10*, *LNCS* 6342, Springer, pp. 83–98, doi:10.1007/978-3-642-15784-4_6.

[16] L. Jacobsen, M. Jacobsen, M.H. Møller & J. Srba (2011): *Verification of Timed-Arc Petri Nets*. In: *Proc. of (SOFSEM'11)*, *LNCS* 6543, Springer-Verlag, pp. 46–72, doi:10.1007/978-3-642-18381-2_4.

[17] A.P. Marques, A.P. Ravn, J. Srba & S. Vighio (2012): *Model-checking web services business activity protocols*. *International Journal on Software Tools for Technology Transfer*, pp. 1–23, doi:10.1007/s10009-012-0231-4.

[18] P. M. Merlin (1974): *A Study of the Recoverability of Computing Systems*. Ph.D. thesis, University of California, Irvine.

[19] E. Newcomer & I. Robinson (chairs) (2009): *Web Services Business Activity (WS-BusinessActivity) Version 1.2*. `http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-os/wstx-wsba-1.2-spec-os.html`.

[20] P. Pettersson (1999): *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*. Ph.D. thesis, Uppsala Univ.

[21] V. V. Ruiz, F. C. Gomez & D. de Frutos-Escrig (1999): *On Non-Decidability of Reachability for Timed-Arc Petri Nets*. In: *Proc. of PNPM'99*, IEEE Computer Society, pp. 188–196.

[22] J. Srba (2005): *Timed-Arc Petri Nets vs. Networks of Timed Automata*. In: *Proc. of ICATPN'05*, *LNCS* 3536, Springer, pp. 385–402, doi:10.1007/11494744_22.

[23] J. Srba (2008): *Comparing the Expressiveness of Timed Automata and Timed Extensions of Petri Nets*. In: *Proc. of FORMATS'08*, *LNCS* 5215, pp. 15–32, doi:10.1007/978-3-540-85778-5_3.