# Verification of Liveness Properties on Closed Timed-Arc Petri Nets*

Mathias Andersen, Heine Gatten Larsen, Jiří Srba, Mathias Grund Sørensen, and Jakob Haahr Taankvist

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark

**Abstract.** Verification of closed timed models by explicit state-space exploration methods is an alternative to the wide-spread symbolic techniques based on difference bound matrices (DBMs). A few experiments found in the literature confirm that for the reachability analysis of timed automata explicit techniques can compete with DBM-based algorithms, at least for situations where the constants used in the models are relatively small. To the best of our knowledge, the explicit methods have not yet been employed in the verification of liveness properties in Petri net models extended with time. We present an algorithm for liveness analysis of closed Timed-Arc Petri Nets (TAPN) extended with weights, transport arcs, inhibitor arcs and age invariants and prove its correctness. The algorithm computes optimized maximum constants for each place in the net that bound the size of the reachable state-space. We document the efficiency of the algorithm by experiments comparing its performance with the state-of-the-art model checker UPPAAL.

## 1 Introduction

TAPAAL [7] is a an efficient, open-source tool for modelling and verification of Timed-Arc Petri Nets (TAPN) extended with transport/inhibitor arcs and age invariants. The timing information (age) is attached to tokens and intervals on input arcs restrict the ages of tokens suitable for transition firing. The verification techniques implemented in the tool include four different translations to UPPAAL timed automata [11], supporting both reachability and liveness properties, and its own verification engine for reachability analysis. The actual verification in any of those approaches rely on searching the abstracted state-space represented via zones and using the data structure called Difference Bound Matrix (DBM) [8].

Unfortunately, for the verification of liveness questions, neither of the methods return error traces (loops in this case) with concrete time delays and not all requested features, like weighted arcs, are currently supported. As in the general case with both open and closed intervals the concrete error traces do not necessarily form a finite loop, we restrict ourselves to the large and practically relevant

* This work is partially supported by the research center IDEA4CPS.

A. Kučera et al. (Eds.): MEMICS 2012, LNCS 7721, pp. 69–81, 2013.
© Springer-Verlag Berlin Heidelberg 2013

subclass of TAPNs with only closed intervals. It is a folklore result that the continuous and discrete-time semantics coincide on the class of closed systems (see e.g. [6]). In nowadays tools the discretization is not sufficiently exploited, perhaps due to its simplicity as remarked by Lamport [12]. Nevertheless, a few existing studies show that discretization of the state-space may be beneficial [6,14,12], at least in the situations with sufficiently small constants that appear in the model.

We suggest an efficient algorithm for verification of liveness properties on closed TAPNs extended with weighted transport/inhibitor arcs and age invariants. The main contributions include a detailed analysis of the maximum constants individually computed for each place in the net, the complete proof of soundness and completeness of the proposed algorithm and last but not least an efficient `C++` implementation and its full integration into the model checker TAPAAL. The efficiency is documented by experiments ranging from standard academic examples for testing the performance of tools like Fischer's protocol for mutual exclusion to larger case-studies from the health-care domain. We compare the CPU-time performance of our discretized algorithm with the efficient DBM-based engines, including the state-of-the-art model checker UPPAAL [1]. The main conclusion is that our algorithm can outperform the DBM-based methods as long as the constants in the model are not too large. Moreover, the discrete method scales very well in the size of the problems, allowing us to solve problems with constants that grow more than linearly while increasing the problem size. As a bonus, our algorithm always returns loop-like counter examples with concrete time delays, a feature that allows the user to easily debug possible design flaws in the models.

**Related Work.** Lamport [12], Bozga et al. [6], Beyer [4,3] and Popova-Zeugmann [14] present different methods for discrete model checking of timed systems. The first three papers present explicit methods for the model of timed automata, while the third one studies discretization for Time Petri Nets (TPN), a Petri net model that is substantially different from ours (timing information is attached to transitions and not to tokens like in TAPNs). While reachability is in general undecidable for TAPNs [16], a time-bounded reachability for TAPNs with discrete semantics was shown decidable in [15]. The technique, however, does not apply for verification of liveness properties because we search here for the presence of an infinite execution satisfying certain invariant properties and such executions are often time-diverging. Our liveness algorithm is instead parameterized by a number $k$ allowing us to explore markings with at most $k$ tokens (after the removal of dead tokens that cannot be used for transition firing). In case of bounded nets it always provides conclusive answers while for unbounded nets (where the liveness problem is undecidable) the answer is conclusive only in the cases where a loop (counter example) can be found among markings with at most $k$ tokens (the number $k$ is a part of the input).

To the best of our knowledge, there are no published experiments for discrete verification of liveness properties on TAPNs, moreover extended with the additional modelling features that require a nontrivial static analysis in order to

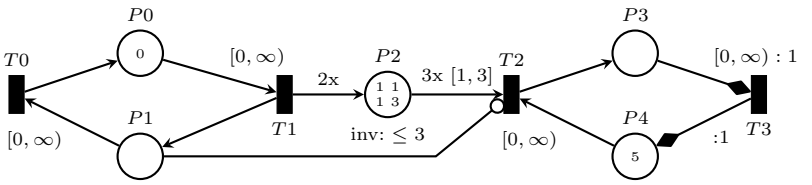


**Fig. 1.** Producer-Consumer Example

minimize the size of maximum constants relative to the individual places in the net. We assess the performance of our approach by performing a comparison against the state-of-the-art model checker UPPAAL and the results are encouraging as documented in Section 5.

## 2 Timed-Arc Petri Nets

Let us first informally introduce the TAPN model. Figure 1 shows an example of a producer-consumer system. The circles represent places and rectangles represent transitions. A marking of the net is given by the distribution of timed tokens; in our case there is one token of age 0 in $P0$, three tokens of age 1 and one of age 3 in $P2$ and one token of age 5 in $P4$.

Places and transitions are connected by arcs and input arcs to transitions are labelled by time intervals. The arc from $T1$ to $P2$ has the weight 2, denoted by 2x, meaning that two tokens will be produced by firing the transition. Similarly the arc from $P2$ to $T2$ has the weight 3, meaning that three tokens of age between 1 and 3 must be consumed when firing $T2$, while at the same time there may not be any token in place $P1$ (denoted by the inhibitor arc with the circle head). In our example the transition $T2$ can fire, consuming three tokens from the place $P2$ (these can be either $\{1, 1, 1\}$ or $\{1, 1, 3\}$) and one token from place $P4$, while depositing a new token of age 0 to the place $P3$. The pair of arcs from $P3$ to $P4$ with a diamond head are called transport arcs and they always come in pairs (in our example with the index :1). They behave like normal arcs but when a token is consumed in $P3$ and produced to $P4$, its age is preserved. Places can also have age invariants like the one denoted by "inv: $\leq 3$" in the place $P2$. This restricts the maximum age of tokens present in such places. In our example, there is already a token of age 3 in $P2$, meaning that we cannot wait any more and are without any delay forced to fire some transition.

Let us now give a formal definition of the TAPN model. Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}_0^\infty = \mathbb{N}_0 \cup \{\infty\}$. A *discrete timed transition system* (DTTS) is a triple $(S, Act, \rightarrow)$ where $S$ is the set of states, $Act$ is the set of actions and $\rightarrow \subseteq S \times (Act \cup \mathbb{N}_0) \times S$ is the transition relation written as $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$. If $a \in Act$ then we call it a *switch transition*, if $a \in \mathbb{N}_0$ we call it a *delay transition*. By $\rightarrow^*$ we denote the reflexive and transitive closure of the relation $\rightarrow \stackrel{\text{def}}{=} \bigcup_{a \in Act} \xrightarrow{a} \cup \bigcup_{d \in \mathbb{N}_0} \xrightarrow{d}$.

We define the set of *well-formed time intervals* as $\mathcal{I} \stackrel{\text{def}}{=} \{[a,b] \mid a \in \mathbb{N}_0, b \in \mathbb{N}_0^\infty, a \leq b\}$ and a subset of $\mathcal{I}$ used in invariants as $\mathcal{I}^{\text{inv}} = \{[0,b] \mid b \in \mathbb{N}_0^\infty\}$. For an interval $[a,b]$ we define $[a,b]_L = a$ and $[a,b]_R = b$ in order to denote the lower and upper bound of the interval, respectively. Let $maxBound(I)$ denote the largest bound different from infinity in the interval $I$, formally $maxBound([a,b]) = a$ if $b = \infty$, and $maxBound([a,b]) = b$ otherwise.

We can now define the closed TAPN model with weighted arcs.

**Definition 1 (Closed Timed-Arc Petri Net).** *A closed TAPN is an 8-tuple* $N = (P, T, IA, OA, g, w, Type, I)$ *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,
- $g : IA \to \mathcal{I}$ *is a* time constraint function *assigning guards to input arcs,*
- $w : IA \cup OA \to \mathbb{N}$ *is a function assigning* weights *to input and output arcs,*
- $Type : IA \cup OA \to \boldsymbol{Types}$ *is a* type function *assigning a type to all arcs, where* $\boldsymbol{Types} = \{Normal, Inhib\} \cup \{Transport_j \mid i \in \mathbb{N}\}$ *such that*
  - *if* $Type(a) = Inhib$ *then* $a \in IA$,
  - *if* $Type((p,t)) = Transport_j$ *for some* $(p,t) \in IA$ *then there is exactly one* $(t,p') \in OA$ *such that* $Type((t,p')) = Transport_j$ *and* $w((p,t)) = w((t,p'))$,
  - *if* $Type((t,p')) = Transport_j$ *for some* $(t,p') \in OA$ *then there is exactly one* $(p,t) \in IA$ *such that* $Type((p,t)) = Transport_j$ *and* $w((p,t)) = w((t,p'))$,
- $I : P \to \mathcal{I}^{inv}$ *is a function assigning* age invariants *to places.*

The preset of input places of a transition $t \in T$ is defined as ${}^\bullet t = \{p \in P \mid (p,t) \in IA, Type((p,t)) \neq Inhib\}$. Similarly, the postset of output places of $t$ is defined as $t^\bullet = \{p \in P \mid (t,p) \in OA\}$.

Let $N = (P, T, IA, OA, g, w, Type, I)$ be a TAPN and let $\mathcal{B}(\mathbb{N}_0)$ be the set of all finite multisets over $\mathbb{N}_0$. A *marking* M on N is a function $M : P \longrightarrow \mathcal{B}(\mathbb{N}_0)$ where for every place $p \in P$ and every token $x \in M(p)$ we have $x \in I(p)$. The set of all markings over N is denoted by $\mathcal{M}(N)$.

We use the notation $(p,x)$ to denote a token at a place $p$ with the age $x \in \mathbb{N}_0$. We write $M = \{(p_1,x_1),(p_2,x_2),\ldots,(p_n,x_n)\}$ for a marking with $n$ tokens of ages $x_i$ located at places $p_i$ and we define $size(M) = \sum_{p \in P} |M(p)|$. A marked TAPN $(N, M_0)$ is a TAPN N together with an initial marking $M_0$ with all tokens of age 0.

**Definition 2 (Enabledness).** *Let* $N = (P, T, IA, OA, g, w, Type, I)$ *be a TAPN. We say that a transition* $t \in T$ *is* enabled *in a marking* $M$ *by the multisets of tokens* $In = \{(p, x_p^1), (p, x_p^2), \ldots, (p, x_p^{w((p,t))}) \mid p \in {}^\bullet t\} \subseteq M$ *and* $Out = \{(p', x_{p'}^1), (p', x_{p'}^2), \ldots, (p', x_{p'}^{w((t,p'))}) \mid p' \in t^\bullet\}$ *if*

– *for all input arcs except the inhibitor arcs the tokens from In satisfy the age guards of the arcs, i.e.*

$$\forall (p,t) \in IA.\mathit{Type}((p,t)) \neq \mathit{Inhib} \Rightarrow x_p^i \in g((p,t)) \text{ for } 1 \leq i \leq w((p,t))$$

– *for any inhibitor arc pointing from a place p to the transition t, the number of tokens in p satisfying the guard is smaller than the weight of the arc, i.e.*

$$\forall (p,t) \in IA.\mathit{Type}((p,t)) = \mathit{Inhib} \Rightarrow |\{x \in M(p) \mid x \in g((p,t))\}| < w((p,t))$$

– *for all input arcs and output arcs which constitute a transport arc the age of the input token must be equal to the age of the output token and satisfy the invariant of the output place, i.e.*

$$\begin{aligned}\forall (p,t) \in IA. \forall (t,p') \in OA.\mathit{Type}((p,t)) = \mathit{Type}((t,p')) = \mathit{Transport}_j \\ \Rightarrow \left(x_p^i = x_{p'}^i \wedge x_{p'}^i \in I(p')\right) \text{ for } 1 \leq i \leq w((p,t)).\end{aligned}$$

– *for all output arcs that are not part of a transport arc the age of the output token is 0, i.e.*

$$\forall (t,p') \in OA.\mathit{Type}((t,p') = \mathit{Normal} \Rightarrow x_{p'}^i = 0 \text{ for } 1 \leq i \leq w((p,t)).$$

In Figure 1 the transition $T2$ is enabled by $In = \{(P2,1),(P2,1),(P2,1),(P4,5)\}$ and $Out = \{(P3,0)\}$. As the tokens in the place $P2$ have different ages, $T2$ is also enabled by an alternative multiset of tokens $In = \{(P2,1),(P2,1),(P2,3),(P4,5)\}$.

A given TAPN $N = (P, T, IA, OA, g, w, \mathit{Type}, I)$ defines a DTTS $T(N) \stackrel{\text{def}}{=} (\mathcal{M}(N), T, \rightarrow)$ where states are the markings and the transitions are as follows.

– If $t \in T$ is enabled in a marking $M$ by the multisets of tokens $In$ and $Out$ then $t$ can be *fire* and produce the marking $M' = (M \setminus In) \uplus Out$ where $\uplus$ is the multiset sum operator and $\setminus$ is the multiset difference operator; we write $M \xrightarrow{t} M'$ for this switch transition.
– A *time delay* $d \in \mathbb{N}$ is allowed in M if $(x + d) \in Inv(p)$ for all $p \in P$ and all $x \in M(p)$, i.e. by delaying $d$ time units no token violates any of the age invariants. By delaying $d$ time units in M we reach the marking $M'$ defined as $M'(p) = \{x + d \mid x \in M(p)\}$ for all $p \in P$; we write $M \xrightarrow{d} M'$ for this delay transition.

A computation of the net $M_0 \rightarrow M_1 \rightarrow \cdots \rightarrow M_n$ is denoted by $\{M_i\}_{i=0}^n$ and we call it a *run*. If the sequence is infinite, we write $\{M_i\}_{i \geq 0}$.

### 2.1 Liveness Verification Problem

The liveness question asks about the existence of a maximal run where every marking satisfies some proposition referring to the distribution of tokens. For that purpose let the set of propositions $\Phi$ be given by the abstract syntax $\varphi ::=$

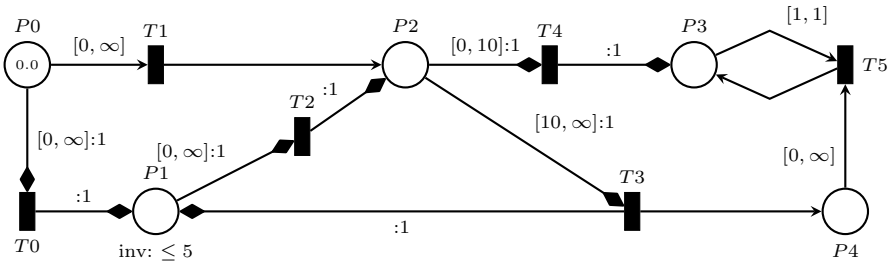


**Fig. 2.** Example of TAPN

$p \bowtie n \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi$ where $\bowtie \in \{\leq, <, =, \neq, \geq, >\}$, $p \in P$ and $n \in \mathbb{N}_0$. The satisfaction relation $M \models \varphi$ is defined in the expected way where $M \models p \bowtie n$ iff $|M(p)| \bowtie n$.

Given a TAPN $(N, M_0)$, a *maximal run* is either an infinite run $\{M_i\}_{i \geq 0}$ or a finite run $\{M_i\}_{i=0}^{n}$ with $M_n \nrightarrow$, meaning that $M_n$ does not allow for any switch or positive-delay transition. A maximal run (finite or infinite) is denoted by $\{M_i\}$.

**Definition 3 (The Liveness Problem ($EG\varphi$)).** *Given a marked TAPN $(N, M_0)$ and a proposition $\varphi \in \Phi$, the liveness problem is to decide whether there is a maximal run $\{M_i\}$ starting in $M_0$ such that $M_i \models \varphi$ for all $i$.*

We can define the standard dual operator $AF$ by $AF\varphi \stackrel{\text{def}}{=} \neg EG\neg\varphi$, meaning that eventually the property $\varphi$ will be satisfied on any execution of the net.

## 3 State-Space Reduction

The state-space of TAPNs is infinite in two dimensions: the number of tokens can be unbounded and the ages of tokens range over natural numbers. Indeed, the model (extended with inhibitor arcs and age invariants) has the full Turing power (see e.g. [16,10]). In order to enable automatic verification, we restrict ourselves to bounded TAPNs where the maximum number of tokens in any reachable marking is a priori bounded by some constant $k$. For restricting the ages of tokens, we do not need to remember the concrete ages of tokens in a place that are older than the maximum constant relevant for that place. This idea was suggested in [16,9] for the basic TAPN model without any additional features. We shall now refine the technique for the more general class of extended TAPNs that contain age invariants, transport and inhibitor arcs and we further enhance it with the reduction of dead tokens in order to optimize its performance.

To motivate the technical definitions that follow, let us consider the net in Figure 2. Note that in place $P3$ the relevant ages of tokens are 0 and 1. Any token of age 2 or more cannot be used for transition firing and can be safely removed from the net. We shall call $P3$ the *dead-token* place with the maximum constant 1. Any place that contains an invariant, like $P1$ in our example, will fall into the category *invariant* places and the maximum constant will be the

$$C_{arc}((p,t)) = \begin{cases} min([I(p')]_R, [g((p,t))]_R) & \text{if } \mathit{Type}((p,t)) = \mathit{Transport}_j, \\ & \mathit{Type}((t,p')) = \mathit{Transport}_j, \text{ and} \\ & I(p') \neq [0,\infty] \\ \mathit{maxBound}(g((p,t))) & \text{otherwise} \end{cases} \quad (1)$$

$$C_{place}(p) = \begin{cases} [I(p)]_R & \text{if } [I(p)]_R \neq \infty \\ -1 & \text{if } I(p) = [0,\infty] \text{ and} \\ & \forall (p,t) \in IA.(g((p,t)) = [0,\infty] \\ \max_{(p,t)\in IA}(C_{arc}((p,t))) & \text{otherwise.} \end{cases} \quad (2)$$

**Fig. 3.** Definitions of $C_{arc}$ and $C_{place}$

upper-bound of the respective invariant. Clearly, no tokens can be older that this constant but at the same time we may not just remove the tokens that cannot be used for any transition firing as they could restrict delay transitions in the future. The remaining places are called *standard* places meaning that instead of the ages of tokens that exceed the maximum constant for a given place, we only need to remember how many there are, but not their exact ages. For example in the place $P0$ all outgoing arcs have the guard $[0,\infty]$, so it may look like that we only need to remember the number of tokens, not their ages. Indeed, if there were no transport arcs this would be the case. However, in our example the pair of transport arcs moving tokens to $P1$ increase the maximum constant for the place $P0$ to 5 as the concrete age is relevant up to this number in order to avoid breaking of the age invariant in $P1$. In general, there might be a series of transport arcs that can influence the maximum constant for a place and we show how to optimize such constants to be as small as possible while still preserving the liveness property we want to verify.

We start by the definition of *causality* function. The causality function finds the causality set of places that are linked with the place $p$ by a chain of transport arcs with the right endpoints of the guard intervals equal to $\infty$.

Let $p \in P$. The set $cau(p)$ is the smallest set of places such that

- $p \in cau(p)$, and
- if $p' \in cau(p)$ and $(p',t) \in IA, (t,p'') \in OA$ such that $\mathit{Type}(p',t) = \mathit{Transport}_j$, $\mathit{Type}(t,p'') = \mathit{Transport}_j$ with $[g((p',t))]_R = \infty$ and $I(p') = \infty$ then $p'' \in cau(p)$.

In the net from Figure 2 we get that for example $cau(P0) = \{P0, P1\}$, $cau(P1) = \{P1\}$ and $cau(P2) = \{P2, P1\}$.

Next we define the maximum relevant constants for input arcs by Equation (1) in Figure 3 as a function $C_{arc} : IA \to \mathbb{N}_0$. The first case deals with the situation when the arc is a transport arc that moves tokens to a place with a nontrivial age invariant; here it is enough to consider the minimum of the invariant upper-bound and the largest constant in the guard different from infinity. If this is not the case, we consider just the maximum bound in the guard.

| Arc | Type | $C_{arc}$ |
|---|---|---|
| $P0 \to T0$ | $Transport_1$ | 5 |
| $P0 \to T1$ | $Normal$ | 0 |
| $P1 \to T2$ | $Transport_1$ | 0 |
| $P2 \to T3$ | $Transport_1$ | 10 |
| $P2 \to T4$ | $Transport_1$ | 10 |

| Place | $C_{place}$ | $C_{max}$ | $cat$ |
|---|---|---|---|
| $P0$ | $-1$ | 5 | $Std$ |
| $P1$ | 5 | 5 | $Inv$ |
| $P2$ | 10 | 10 | $Std$ |
| $P3$ | 1 | 1 | $Dead$ |
| $P4$ | $-1$ | $-1$ | $Std$ |

**Fig. 4.** Calculation of $C_{arc}$, $C_{place}$, $C_{max}$ and $cat$ for the TAPN in Figure 2

The constant of a place (without considering any causality) is defined by Equation (2) in Figure 3 as a function $C_{place} : P \to \mathbb{N}_0 \cup \{-1\}$. The constant is either the upper-bound of a nontrivial age invariant in the place, or $-1$ if all arcs from $p$ only have trivial guards; in this case we do not care about the ages of the tokens in $p$. Otherwise the constant for $p$ is the largest constant of any arc starting at $p$.

We are now ready to divide places into three categories and compute the maximum relevant constants taking into account the causality set of places. In liveness verification, the query will also influence the category of places, so we consider the function $Places : \Phi \to \mathcal{P}(P)$ that for a given proposition $\varphi$ returns the set of places that syntactically appear in $\varphi$. We can now calculate the function $C_{max} : P \to \mathbb{N}_0 \cup \{-1\}$ returning the maximum constant for each place (taking into account also the transport arcs) and the function $cat : P \to \{Inv, Dead, Std\}$ returning the category for each place $p \in P$ as follows.

- If $I(p) \neq [0, \infty]$ then $C_{max}(p) = [I(p)]_R$ and $cat(p) = Inv$.
- Otherwise $C_{max}(p) = \max\{C_{place}(p') \mid p' \in cau(p)\}$ and if either
  - (i) there is $t \in T$ such that $(p, t) \in IA$ and $Type((p, t)) = Inhib$), or
  - (ii) there is $t \in T$ such that $(p, t) \in IA$ and $[g((p, t))]_R = \infty]$, or
  - (iii) $p \in Places(\varphi)$)

  then $cat(p) = Std$, else $cat(p) = Dead$.

The conditions (i)–(iii) list all situations where we are not allowed to remove tokens above the maximum constant as the concrete number of these tokens is relevant for the behaviour of the net or for the the proposition $\varphi$. An example of the calculation of $C_{max}$ and $cat$ is given in Figure 4, assuming $Places(\varphi) = \{P1\}$.

### 3.1 Bounded Marking Equivalence

Given the maximum constants and categories of places, we can now define an equivalence relation on markings that will have a finite number of equivalence classes and can be used in the liveness checking algorithm.

Let $C_{max}$ and $cat$ be computed as above and let $M$ be a marking. We split $M$ into two markings $M_>$ and $M_\leq$ as follows: $M_>(p) = \{x \in M(p) \mid x > C_{max}(p)\}$ and $M_\leq(p) = \{x \in M(p) \mid x \leq C_{max}(p)\}$ for all places $p \in P$. Clearly, $M = M_> \uplus M_\leq$.

**Definition 4 (Bounded Marking Equivalence).** *Let $M$ and $M'$ be markings on a TAPN $N$. We say that $M$ and $M'$ are equivalent, written $M \equiv M'$, if*

- $M_{\leq}(p) = M'_{\leq}(p)$ *for all* $p \in P$, *and*
- $|M_{>}(p)| = |M'_{>}(p)|$ *for all* $p \in P$ *where* $cat(p) = Std$.

The equivalence relation implies that in *Dead* places we do not care about the tokens with ages greater than $C_{max}$ and that in *Std* places we do not care about tokens with ages greater than $C_{max}$, as long as there are equally many of them in both markings. An important correctness argument is the fact that that the relation $\equiv$ is a timed bisimulation where delays on one side and matched by exactly the same delays on the other side (see e.g. [13]). The proof is done by a detailed case analysis and can be found in the full version of the paper.

**Theorem 1.** *The relation $\equiv$ is a timed bisimulation.*

In order to calculate a representative marking for each $\equiv$-equivalence class, we define the function *cut* and present Lemma 1 that is proved in the full version of the paper.

**Definition 5 (Cut).** *The function $cut : \mathcal{M}(N) \to \mathcal{M}(N)$ is given by*

$$cut(M)(p) = \begin{cases} M_{\leq}(p) & \text{if } cat(p) \in \{Inv, Dead\} \\ M_{\leq}(p) \uplus \{ \underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_{>}(p)| \text{ times}} \} & \text{if } cat(p)=Std \end{cases}$$

*for all $p \in P$. We call the marking $cut(M)$* canonical.

**Lemma 1 (Properties of Canonical Markings)**

1. *For any marking $M$ we have $M \equiv cut(M)$.*
2. *Given two markings $M_1$ and $M_2$ if $M_1 \equiv M_2$ then $cut(M_1) = cut(M_2)$.*
3. *Let $M$ be a marking and $\varphi \in \Phi$ be a proposition then $M \models \varphi$ iff $cut(M) \models \varphi$.*

## 4 Liveness Algorithm

We can now present Algorithm 1 answering the liveness verification problem. It is essentially a depth-first search algorithm where the *Waiting* stack stores the currently unexplored successors that satisfy the invariant property $\varphi$. In the *Trace* stack we keep track of the run from the initial marking to the currently explored marking. A counter recording the number of unexplored successors for each marking on the *Trace* stack is used for the coordination between the *Trace* and *Waiting* stacks. The main loop contains a boolean variable indicating whether the current marking is the end of a maximal run (in case no further successors exist). If this is the case, the algorithm terminates as a maximal run satisfying $\varphi$ has been found. Otherwise new canonical successors (by transition firing and one-unit delay) are added by calling the function $AddToPW$, making

```
1  input: A TAPN (N, M_0), proposition φ ∈ Φ and k ∈ ℕ s.t. size(cut(M_0)) ≤ k.
2  output: True if there is a maximal run {M_i} s.t. M_i ⊨ φ and
   size(cut(M_i)) ≤ k, false otherwise.
3  begin
4      Passed := ∅; Waiting.InitStack(); Trace.InitStack(); M'_0 := cut(M_0);
5      if M'_0 ⊨ φ then
6          Waiting.push(M'_0);
7      while ¬Waiting.isEmpty() do
8          M := Waiting.pop();
9          if M ∉ Passed then
10             Passed := Passed ∪ {M}; M.successors:=0;
11             Trace.push(M); endOfMaxRun := true;
12             foreach M' s.t. M --t--> M' do
13                 AddToPW(M,M'); endOfMaxRun := false;
14             if min_{(p,x)∈M}([I(p)]_R − x) > 0 then
15                 AddToPW(M,M') where M --1--> M'; endOfMaxRun := false;
16             if endOfMaxRun then
17                 return true      /* terminate and return the Trace stack */ ;
18         else
19             Trace.top().successors−−
20         while ¬Trace.isEmpty() ∧ Trace.top().successors = 0 do
21             Trace.pop();
22             if Trace.isEmpty() then
23                 return false      /* terminate the whole algorithm *.;
24             Trace.top().successors−−;
25     return false;
26 AddToPW(M,M'): begin
27     M'' := cut(M');
28     if M'' ∈ Trace then
29         return true      /* terminate and return the loop on the Trace stack */;
30     if M'' ∉ Passed ∧ M'' ⊨ φ ∧ size(M'') ≤ k  then
31         Waiting.push(M'');
32         M.successors++;
```

**Algorithm 1.** Liveness algorithm

sure that only markings that satisfy $\varphi$ are added to the *Waiting* list. The function also checks for the presence of a loop on the *Trace* stack, in which case the algorithm terminates and returns true. A bound $k$ is also an input to the algorithm, making sure that only canonical markings with no more than $k$ tokens are explored during the search. If the net is $k$-bounded, this has no effect on the actual search. For unbounded nets, our algorithm still terminates and provides a suitable under-approximation of the net behaviour, giving conclusive answers if a loop is found and inconclusive answers otherwise.

**Table 1.** Fischer's protocol scaled by the number of processes (rows) and the size of maximum constant (columns). First line is a native UPPAAL model, second line is the fastest translation to timed automata and using the UPPAAL engine, and third line is our discrete TAPAAL engine. The symbol ◷ stands for more than 900 seconds.

| Processes \ Constants | 3 | 5 | 7 | 9 | 11 | 13 | 15 |
|---|---|---|---|---|---|---|---|
| 5 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| | 0.1 | 0.1 | 0.3 | 0.7 | 1.8 | 3.7 | 7.9 |
| 6 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 |
| | **0.1** | **0.1** | 0.5 | 1.8 | 5.3 | 13.3 | 29.6 |
| 7 | 4.6 | 4.6 | 4.6 | 4.6 | 4.6 | 4.6 | 4.6 |
| | 47.5 | 47.2 | 47.0 | 47.1 | 47.2 | 47.4 | 47.1 |
| | **0.1** | **0.2** | **1.1** | **4.5** | 14.4 | 40.7 | 99.3 |
| 8 | 422.5 | 422.6 | 421.5 | 422.4 | 421.9 | 422.1 | 422.3 |
| | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ |
| | **0.1** | **0.4** | **2.4** | **10.5** | **37.8** | **115.2** | **309.8** |
| 9 | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ |
| | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ |
| | **0.1** | **0.7** | **4.5** | **22.4** | **90.5** | **300.4** | **888.2** |
| 10 | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ |
| | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ | ◷ |
| | **0.1** | **1.1** | **8.2** | **45.9** | **202.2** | **733.5** | ◷ |

**Table 2.** Blood transfusion case study scaled by the number of patients; time is seconds

| Patients | Translations | TAPAAL |
|---|---|---|
| 1 | 0.11 | 0.04 |
| 2 | 28.08 | 0.93 |
| 3 | >5400.00 | 30.47 |
| 4 | >5400.00 | 1072.50 |

**Theorem 2 (Correctness).** *Let TAPN $(N, M_0)$ be a closed TAPN, $\varphi \in \Phi$ a proposition and $k \in \mathbb{N}$ a number such that $size(cut(M_0)) \leq k$. Algorithm 1 terminates, and it returns true if there is a maximal run $\{M_i\}$ such that $M_i \models \varphi$ and $size(cut(M_i)) \leq k$ and false otherwise.*

*Proof (sketch).* Termination follows from the fact that we only store markings after applying the function *cut*, giving us together with at most $k$ tokens in the net a finite state-space. The soundness and completeness part of Theorem 2 rely on Lemma 1 and Theorem 1 and details are given the full version of the paper. □

## 5 Experiments

The liveness algorithm has been implemented and fully integrated into the verification tool TAPAAL [7] and it can be downloaded (as a beta-release) from

`http://www.tapaal.net`. We performed a number of experiments[1] and due to the space limitation mention only two of them. The results of verification of Fischer's algorithm for mutual exclusion are given in Table 1. We asked here an EG query checking whether there is an infinite schedule allowing us to repeatedly enter the critical section within a given time interval. The query is not satisfied and hence the whole state-space where the proposition holds is searched. The table shows the verification times for a native UPPAAL model of the protocol (first line), the best time for a translation (see [7] for details) to timed automata and then using the UPPAAL engine (second line) and our discretized algorithm (third line). The gray cells mark the experiments where our method was the fastest one. The reader can observe that the DBM-based methods in the first two lines are immune to scaling of constants. On the other hand, our algorithm scales significantly better with increasing the number of processes. Hence for larger instances, we can handle larger and larger constants while still outperforming the DBM-based methods. In fact, the size of the constants we can deal with for the given time limit grows more than linearly as we increase the number of processes. We have observed similar behaviour in other case studies too, like e.g. in the Lynch-Shavit protocol that is presented in the full version of the paper.

In order to test the performance on a realistic case-study, we verified soundness (AF query) of a blood transfusion medical workflow (details can be found in [2]) where the maximum constant is of size 90 and it considerably outperforms the translation approach verified via UPPAAL engine. Results are given in Table 2 and we compare our engine with the fastest translation to UPPAAL timed automata.

## 6 Conclusion

We presented a discrete algorithm for verification of liveness properties on extended timed-arc Petri nets and provided its implementation and integration into the model checker TAPAAL. The main technical contribution is the partitioning of the places in the net to three categories and an optimized computation of the individual maximum constants, allowing us to design an efficient loop detection algorithm based on depth-first search. We proved the algorithm correct and demonstrated on several examples its applicability as an alternative to DBM-based search algorithms. The techniques can be easily adapted to work also for reachability analysis.

Our approach is well suited for larger models with relatively small constants. Due to an on-the-fly removal of dead tokens that appear in the net, we were able to successfully verify models that are in general unbounded and where DBM-based methods give inconclusive answers (for example in case of the Alternating Bit Protocol (ABP) with perfect communication channels presented as the standard example in the TAPAAL distribution). In the future work we shall focus

[1] We report here the data obtained on MacBook Pro 2.7GHz INTEL Core i7 with 8 GB RAM and 64-bit versions of UPPAAL and TAPAAL.

on space-optimization of the proposed technique, on a symbolic computation of the delay operator and on comparing the method to BDD-based state space exploration (as exploited e.g. in the tool Rabbit [5]).

## References


1. Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: QEST 2006, pp. 125–126 (September 2006)
2. Bertolini, C., Liu, Z., Srba, J.: Verification of timed healthcare workflows using component timed-arc Petri nets. In: FHIES 2012. Springer (to appear, 2012)
3. Beyer, D.: Efficient Reachability Analysis and Refinement Checking of Timed Automata Using BDDs. In: Margaria, T., Melham, T.F. (eds.) CHARME 2001. LNCS, vol. 2144, pp. 86–91. Springer, Heidelberg (2001)
4. Beyer, D.: Improvements in BDD-Based Reachability Analysis of Timed Automata. In: Oliveira, J.N., Zave, P. (eds.) FME 2001. LNCS, vol. 2021, pp. 318–343. Springer, Heidelberg (2001)
5. Beyer, D., Lewerentz, C., Noack, A.: Rabbit: A Tool for BDD-Based Verification of Real-Time Systems. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 122–125. Springer, Heidelberg (2003)
6. Bozga, M., Maler, O., Tripakis, S.: Efficient Verification of Timed Automata Using Dense and Discrete Time Semantics. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 125–141. Springer, Heidelberg (1999)
7. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 492–497. Springer, Heidelberg (2012)
8. Dill, D.L.: Timing Assumptions and Verification of Finite-state Concurrent Systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
9. Hanisch, H.M.: Analysis of Place/transition Nets with Timed Arcs and its Application to Batch Process Control. In: Ajmone Marsan, M. (ed.) ICATPN 1993. LNCS, vol. 691, pp. 282–299. Springer, Heidelberg (1993)
10. Jacobsen, L., Jacobsen, M., Møller, M.H.: Undecidability of Coverability and Boundedness for Timed-Arc Petri Nets with Invariants. In: Proc. of MEMICS 2009, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2009)
11. Jacobsen, L., Jacobsen, M., Møller, M.H., Srba, J.: Verification of timed-arc Petri nets. In: SOFSEM 2011, pp. 46–72 (2011)
12. Lamport, L.: Real-Time Model Checking Is Really Simple. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 162–175. Springer, Heidelberg (2005)
13. Larsen, K.G., Wang, Y.: Time-abstracted bisimulation: Implicit specifications and decidability. Information and Computation 134(2), 75–101 (1997)
14. Popova-zeugmann, L.: Essential states in time Petri nets. Informatik-Berichte 96 (1998)
15. de Frutos Escrig, D., Ruiz, V.V., Marroquín Alonso, O.: Decidability of Properties of Timed-Arc Petri Nets. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 187–206. Springer, Heidelberg (2000)
16. Ruiz, V.V., Cuartero Gomez, F., de Frutos Escrig, D.: On non-decidability of reachability for timed-arc Petri nets. In: Proceedings of the 8th International Workshop on Petri Net and Performance Models (PNPM 1999), pp. 188–196 (1999)