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 [8] 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 [12], 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) [9].

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 subclass of TAPNs with only closed intervals. It is a folklore result

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

that the continuous and discrete-time semantics coincide on the class of closed systems (see e.g. [7]). In nowadays tools the discretization is not sufficiently exploited, perhaps due to its simplicity as remarked by Lamport [13]. Nevertheless, a few existing studies show that discretization of the state-space may be beneficial [7, 15, 13], 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 [2]. 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 [13], Bozga et al. [7], Beyer [5, 4] and Popova-Zeugmann [15] 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 [17], a time-bounded reachability for TAPNs with discrete semantics was shown decidable in [16]. 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

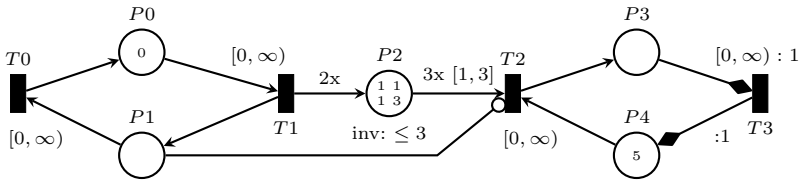

Fig. 1. Producer-Consumer Example

to 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: ≤ 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.*

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

– *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.*

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

– *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.*

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

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

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

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 $N = (P, T, IA, OA, g, w, Type, I)$ 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 Φ be given by the abstract syntax $\varphi ::=$

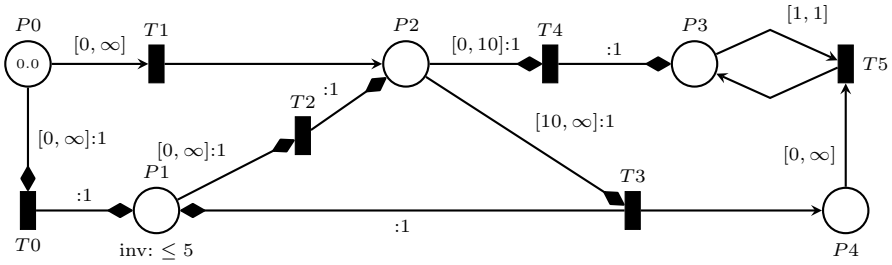

Fig. 2. Example of TAPN

$p \bowtie n \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi$ 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 φ 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. [17, 11]). 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 [17, 10] 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

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

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

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

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 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 ∞.

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

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

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.

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 φ returns the set of places that syntactically appear in φ. 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 φ. 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. [14]). 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 φ. 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

```
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

successors exist). If this is the case, the algorithm terminates as a maximal run satisfying φ has been found. Otherwise new canonical successors (by transition firing and one-unit delay) are added by calling the function *AddToPW*, making sure that only markings that satisfy φ 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.

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 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.

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

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

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 [8] 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 [8] 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 [3]) 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 on space-optimization of the proposed technique, on a symbolic computation

[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.

of the delay operator and on comparing the method to BDD-based state space exploration (as exploited e.g. in the tool Rabbit [6]).

References

1. P.A. Abdulla, J. Deneux, P. Mahata, and A. Nylén. Using forward reachability analysis for verification of timed Petri nets. *Nordic J. of Computing*, 14:1–42, 2007.
2. G. Behrmann, A. David, K.G. Larsen, J. Hakansson, P. Petterson, Wang Yi, and M. Hendriks. Uppaal 4.0. In *QEST'06*, pages 125 –126, sept. 2006.
3. C. Bertolini, Zh. Liu, and J. Srba. Verification of timed healthcare workflows using component timed-arc Petri nets. In *FHIES'12*. Springer-Verlag, 2012. To appear.
4. D. Beyer. Efficient reachability analysis and refinement checking of timed automata using BDDs. In *Proc. of CHARME'01*, volume 2144 of *LNCS*, pages 86–91. Springer-Verlag, 2001.
5. D. Beyer. Improvements in BDD-based reachability analysis of timed automata. In *Proc. of FME'01*, volume 2021 of *LNCS*, pages 318–343. Springer-Verlag, 2001.
6. D. Beyer, C. Lewerentz, and A. Noack. Rabbit: A tool for BDD-based verification of real-time systems. In *Proc. of CAV'03*, volume 2725 of *LNCS*, pages 122–125. Springer-Verlag, 2003.
7. M. Bozga, O. Maler, and S. Tripakis. Efficient verification of timed automata using dense and discrete time semantics. In *CHARME'99*, volume 1703 of *LNCS*, pages 125–141. Springer, 1999.
8. A. David, L. Jacobsen, M. Jacobsen, K.Y. Jørgensen, M.H. Møller, and J. Srba. TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In *TACAS'12*, volume 7214 of *LNCS*, pages 492–497. Springer-Verlag, 2012.
9. D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 197–212. Springer, 1989.
10. H.M. Hanisch. Analysis of place/transition nets with timed arcs and its application to batch process control. In *Application and Theory of Petri Nets 1993*, volume 691 of *LNCS*, pages 282–299. Springer Berlin / Heidelberg, 1993.
11. L. Jacobsen, M. Jacobsen, and M. H. Møller. Undecidability of Coverability and Boundedness for Timed-Arc Petri Nets with Invariants. In *Proc. of MEMICS'09*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009.
12. L. Jacobsen, M. Jacobsen, M.H. Møller, and J. Srba. Verification of timed-arc Petri nets. In *SOFSEM'11*, pages 46–72, 2011.
13. L. Lamport. Real-time model checking is really simple. In Dominique Borrione and Wolfgang Paul, editors, *Correct Hardware Design and Verification Methods*, volume 3725 of *Lecture Notes in Computer Science*, pages 162–175. Springer Berlin / Heidelberg, 2005.
14. K.G. Larsen and Y. Wang. Time-abstracted bisimulation: Implicit specifications and decidability. *Information and Computation*, 134(2):75 – 101, 1997.
15. Louchka Popova-zeugmann. Essential states in time Petri nets. *Informatik-Berichte*, 96, 1998.
16. V.V. Ruiz, D. de Frutos Escrig, and O. Marroquin Alonso. Decidability of properties of timed-arc Petri nets. In *ICATPN'00*, volume 1825 of *LNCS*, pages 187–206. Springer-Verlag, 2000.
17. V.V. Ruiz, F. Cuartero Gomez, and D. de Frutos Escrig. On non-decidability of reachability for timed-arc Petri nets. In *Proceedings of the 8th International Workshop on Petri Net and Performance Models (PNPM'99)*, pages 188–196, 1999.

A Proofs of Timed Bisimilarity

Before we proceed to Theorem 1, we will prove a technical lemma that will allow us to provide a more structured proof of the theorem.

Lemma 2. *Let M_1, M_1', M_2, M_2' be markings such that $M_1 \equiv M_1'$ and $M_2 \equiv M_2'$. Then $M_1 \uplus M_2 \equiv M_1' \uplus M_2'$. If moreover $M_2 \subseteq M_1$ and $M_2' \subseteq M_1'$, then $M_1 \setminus M_2 \equiv M_1' \setminus M_2'$.*

Proof. Let M_1, M_1', M_2, M_2' be markings as assumed in the lemma. By the definition of the multiset sum operator and the multiset difference operator, we get for all $p \in P$ the following equalities.

$$(M_1 \uplus M_2)_{\leq}(p) = M_{1\leq}(p) \uplus M_{2\leq}(p) = M'_{1\leq}(p) \uplus M'_{2\leq}(p) = (M_1' \uplus M_2')_{\leq}(p)$$

$$(M_1 \setminus M_2)_{\leq}(p) = M_{1\leq}(p) \setminus M_{2\leq}(p) = M'_{1\leq}(p) \setminus M'_{2\leq}(p) = (M_1' \setminus M_2')_{\leq}(p)$$

For all $p \in P$ such that $cat(p) = Std$ we also get the remaining two equations establishing the desired equivalence.

$$\begin{aligned} |(M_1 \uplus M_2)_{>}(p)| &= |M_{1>}(p) \uplus M_{2>}(p)| = |M_{1>}(p)| + |M_{2>}(p)| \\ &= |{M_1}'_{>}(p)| + |M'_{2>}(p)| = |M'_{1>}(p) \uplus M'_{2>}(p)| \\ &= |(M_1' \uplus M_2')_{>}(p)| \end{aligned} \tag{3}$$

$$\begin{aligned} |(M_1 \setminus M_2)_{>}(p)| &= |M_{1>}(p) \setminus M_{2>}(p)| = |M_{1>}(p)| - |M_{2>}(p)| \\ &= |{M_1}'_{>}(p)| - |M'_{2>}(p)| = |M'_{1>}(p) \setminus M'_{2>}(p)| \\ &= |(M_1' \setminus M_2')_{>}(p)| \end{aligned} \tag{4}$$

Note that the requirement $M_2 \subseteq M_1$ and $M_2' \subseteq M_1'$ is necessary for Equation (4) as in equivalent markings the ages of tokens above the maximum constant do not have to match but we want to be sure that all tokens from M_2 and M_2' are indeed removed when the multiset difference operator is applied. □

We can now give a complete proof of the main theorem.

Proof (Theorem 1). Consider the TAPN $N = (P, T, IA, OA, g, w, Type, I)$ and a pair of markings M_1 and M_2 such that $M_1 \equiv M_2$. We want to show that $\equiv$ is a timed bisimulation, in other words, that

1. if $M_1 \xrightarrow{t} M_1'$ then $M_2 \xrightarrow{t} M_2'$ and $M_1' \equiv M_2'$,
2. if $M_1 \xrightarrow{d} M_1'$ then $M_2 \xrightarrow{d} M_2'$ and $M_1' \equiv M_2'$,
3. if $M_2 \xrightarrow{t} M_2'$ then $M_1 \xrightarrow{t} M_1'$ and $M_2' \equiv M_1'$, and
4. if $M_2 \xrightarrow{d} M_2'$ then $M_1 \xrightarrow{d} M_1'$ and $M_2' \equiv M_1'$.

Condition 1. Let a transition $t \in T$ be enabled in the marking M_1 by the sets of tokens $In_1 \subseteq M_1$ and Out_1 according to Definition 2. We shall construct two sets of tokens $In_2 \subseteq M_2$ and Out_2 that enable t in M_2 and argue that

$$M_1' = (M_1 \setminus In_1) \uplus Out_1 \equiv (M_2 \setminus In_2) \uplus Out_2 = M_2' \ .$$

To simplify the proof, we will use Lemma 2 and argue instead for two simpler (but equivalent) claims that $In_1 \equiv In_2$ and $Out_1 \equiv Out_2$. In what follows let $In^{type} = \{(p, x) \in In \mid cat(p) = type\}$ where $type \in \{Inv, Dead, Std\}$.

We shall now present the construction of the multisets In_2 and Out_2 from the given multisets In_1 and Out_1. We start with the definition of $In_2 = In_2^{Inv} \uplus In_2^{Dead} \uplus In_2^{Std}$ and argue that $In_2 \subseteq M_2$.

- Let $In_2^{Inv} := In_1^{Inv}$. Clearly $In_2^{Inv} \equiv In_1^{Inv}$ and all tokens in the invariant places of M_1 are also present in the equivalent marking M_2, giving us that $In_2^{Inv} \subseteq M_2$ by the assumption that $In_1^{Inv} \subseteq In_1 \subseteq M_1$.
- Let $In_2^{Dead} := In_1^{Dead}$ and again clearly $In_2^{Dead} \equiv In_1^{Dead}$. Any token $(p, x) \in In_1^{Dead}$ must satisfy that $x \leq C_{max}(p)$; otherwise the set In_1 would not enable the transition t because the interval on the arc from p to t has an upper-bound that is (by definition of dead-token places) strictly below x. Hence we get $In_2^{Dead} \subseteq M_2$ by the same arguments as in the first case.
- The set In_2^{Std} is now defined by two cases. For every token $(p, x) \in In_1^{Std}$:
  - if $x \leq C_{max}(p)$ we add (p, x) to the multiset In_2^{Std}, and
  - if $x > C_{max}(p)$ we add to In_2^{Std} any token $(p, x') \in M_2$ with $x' > C_{max}(p)$ that has not been added to In_2^{Std} yet. Such tokens can be always found by the fact that $|M_{1>}(p)| = |M_{2>}(p)|$ which follows from $M_1 \equiv M_2$ and from the assumption that $cat(p) = Std$.

  We can conclude that $In_1^{Std} \equiv In_2^{Std}$ and $In_2^{Std} \subseteq M_2$.

We have already argued in the three cases above that $In_2 \subseteq M_2$ and because $In_2^{Inv} \equiv In_1^{Inv}$ and $In_2^{Dead} \equiv In_1^{Dead}$ and $In_2^{Std} \equiv In_1^{Std}$ we get by Lemma 2 that also

$$In_2 = In_2^{Inv} \uplus In_2^{Dead} \uplus In_2^{Std} \equiv In_1^{Inv} \uplus In_1^{Dead} \uplus In_1^{Std} = In_1 \ . \tag{5}$$

Let us now continue with the definition of Out_2 where we want to argue that $Out_2 \equiv Out_1$.

- *Normal output arcs.* For each $(t, p') \in OA$ such that $Type((t, p')) = Normal$ we add to Out_2 as many tokens $(p', 0)$ as is the weight of the arc; formally we add to Out_2 the multiset of tokens $\underbrace{\{(p', 0), (p', 0), \ldots, (p', 0)\}}_{w((t,p'))\text{–times}}$. Clearly, $\{(p', x) \in Out_2 \mid Type((t, p')) = Normal\} \equiv \{(p', x) \in Out_1 \mid Type((t, p')) = Normal\}$ as the newly added tokens are identical to those that are in Out_1.
- *Transport output arcs.* For each output arc $(t, p') \in OA$ where $Type((t, p')) = Transport_j$, there is a unique place $p \in P$ with $Type((p, t)) = Transport_j$ and in In_1 there are the tokens $(p, x_p^i) \in In_1$, $1 \leq i \leq w((p, t))$, that are moved to p'. Now, for each $1 \leq i \leq w((p, t))$, if $x_p^i \leq C_{max}(p)$ then also $(p, x_p^i) \in In_2$

and we add the token (p', y_p^i) where $y_p^i = x_p^i$ into Out_2; if $x_p^i > C_{max}(p)$ then there is a corresponding token $(p, y_p^i) \in In_2$ with $y_p^i > C_{max}(p)$ and we add (p', y_p^i) into Out_2. What remains to be established is that $Out_2 \equiv Out_1$ for the tokens added in this second case. Assume we have added a token (p', y_p^i) into Out_2. If $y_p^i = x_p^i$ then we are done. Otherwise there are two cases according to the category of the place p'.

- If $cat(p') = Inv$ then we know that $x_p^i \leq [I(p')]_R$ giving us that $x_p^i \leq C_{max}(p)$ by the definitions in Figure 3 and the causality function in combination with the definition of C_{max}. This means that by our construction $y_p^i = x_p^i$.
- If $cat(p') = Dead$ or $cat(p') = Std$ then there are two subcases.
  - If $[g((p,t))]_R \neq \infty$ then clearly $x_p^i \leq [g((p,t))]_R \leq C_{max}(p)$ and by our construction we get $y_p^i = x_p^i$.
  - If $[g((p,t))]_R = \infty$ then $C_{max}(p) \geq C_{max}(p')$ because $p' \in cau(p)$ which implies that $cau(p') \subseteq cau(p)$ and from the fact that $C_{max}(p)$ is defined as $\max\{C_{place}(p') \mid p' \in cau(p)\}$. Now if $x_p^i > C_{max}(p) \geq C_{max}(p')$ then also $y_p^i > C_{max}(p) \geq C_{max}(p')$ and we are done also in this case.

Hence we established that $Out_2 \equiv Out_1$ which together with the previously proven fact $In_2 \equiv In_1$ implies by Lemma 2 that

$$M_1' = (M_1 \setminus In_1) \uplus Out_1 \equiv (M_2 \setminus In_2) \uplus Out_2 = M_2' \ .$$

What remains to be verified is that in the marking M_2 the transition t is indeed enabled by the multisets of tokens In_2 and Out_2. We do this by checking the four conditions in Definition 2.

- For every input arc $(p,t) \in IA$ which is not an inhibitor arc, we know that in M_1 the arc's interval is satisfied by the tokens (p, x_p^i), $1 \leq i \leq w((p,t))$, that are part of In_1. We need to check that the corresponding tokens (p, y_p^i), $1 \leq i \leq w((p,t))$, in In_2 also satisfy the interval, using the fact that $In_1 \equiv In_2$. Indeed, if $x_p^i = y_p^i$ then the statement is trivially true. Otherwise we know that both $x_p^i, y_p^i > C_{max}(p)$ and hence the tokens are in particular larger than any integer constant appearing on the interval. This implies that the interval on the arc (p,t) must be open to the right as the token x_p^i fits into the interval by our assumption. Hence y_p^i also fits into the interval.
- For every inhibitor arc $(p,t) \in IA$, we know by the definition of categories of places that $cat(p) \neq Dead$, meaning that both M_1 and M_2 must have the same number of tokens in the place p. Moreover, by the arguments as above, because in M_1 the number of tokens in the interval of the inhibitor arc is less than its weight, so will be the number of tokens in the marking M_2.
- Now for every pair of transport arcs (p,t) and (t,p') we know by the definition of Out_2 that the tokens moved from p to p' preserve their ages. We just need to verify that they also satisfy the invariant in the place p'. If $[I(p)]_R \leq [I(p')]_R$ or $[g(p,t)]_R \leq I(p')$ then this is trivially true. Otherwise by the

definition of C_{max} and by the definitions in Figure 3 we have $C_{max}(p) \geq I(p')$ because of the pair of transport arcs (p, t) and (t, p'). This means that the ages of tokens in In_1 and In_2 that travel across this pair of transport arcs will be the same in both cases and we are done also with this case.
- The last condition that the tokens in Out_2 produced by normal arcs are of age 0 is true by the construction of Out_2.

Condition 2. Assume that $M_1 \xrightarrow{d} M_1'$. This means that for any place p of type Inv also $x + d \in I(p)$ for any token $x \in M_1(p)$. Whenever $cat(p) = Inv$, all tokens in the place p are of age at most $C_{max}(p)$ and thus $M_1(p) = M_2(p)$ for any such p. This means that $M_2 \xrightarrow{d} M_2'$. Moreover, by aging tokens in the two markings M_1 and M_2 by equal delays cannot break any of the two requirements of Definition 4. Therefore $M_1' \equiv M_2'$.

Conditions 3 and 4. The proofs of these two conditions are symmetric to the two cases above. □

B Proofs for Canonical Representative

B.1 Lemma 1.1

Proof (Lemma 1.1). We have to show that the two properties in Definition 4 hold for the pair of markings M and $cut(M)$.

- $cut(M)_{\leq}(p) = M_{\leq}(p)$ for all $p \in P$:
  - If $cat(p) \in \{Inv, Dead\}$ then $cut(M)(p) = M_{\leq}(p)$ by definition.
  - If $cat(p) = Std$ then

$$cut(M)(p) = M_{\leq}(p) \uplus \{ \underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_{>}(p)| \text{ times}} \}$$

    by definition and the tokens we add to the right are older than $C_{max}(p)$ thus $cut(M)_{\leq}(p) = M_{\leq}(p)$.
- $|cut(M)_{>}(p)| = |M_{>}(p)|$ for $p \in P$ such that $cat(p) = Std$:
  - $cut(M)(p) = M_{\leq}(p) \uplus \{ \underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_{>}(p)| \text{ times}} \}$ by definition and thus $cut(M)_{>}(p) = \{ \underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_{>}(p)| \text{ times}} \}$ and we conclude that $|cut(M)_{>}(p)| = |M_{>}(p)|$.

□

B.2 Lemma 1.2

Proof (Lemma 1.2). Assume that $M_1 \equiv M_2$. If $cat(p) \in \{Inv, Dead\}$ we have

$$cut(M_1)(p) = M_{1\leq}(p) = M_{2\leq}(p) = cut(M_2)(p).$$

If $cat(p) = Std$ we have

$$\begin{aligned} cut(M_1)(p) &= M_{1\leq}(p) \uplus \{ \underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_{1>}(p)| \text{ times}} \} \\ &= M_{2\leq}(p) \uplus \{ \underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_{2>}(p)| \text{ times}} \} \\ &= cut(M_2)(p). \end{aligned}$$

□

B.3 Lemma 1.3

Proof (Lemma 1.3). Propositions only ask about the number of tokens in different places, not about the ages of the tokens. By Lemma 1 part 1 and Definition 4, the markings M and $cut(M)$ have exactly the same number of tokens at every place, with the exception of dead-token places. Since a dead token place p must satisfy $p \notin Places(\varphi)$, the number of tokens in this place is irrelevant for the proposition. □

C Correctness of the Liveness Algorithm

We shall now argue about the correctness of the liveness algorithm by proving its termination, soundness and completeness.

Lemma 3 (Termination). *Algorithm 1 terminates on any input.*

Proof. The algorithm terminates once the *Waiting* stack gets empty. Observe that only canonical markings of size at most k can be pushed to the *Waiting* stack. There are only finitely many such canonical markings. The termination now follows from the fact that in every iteration of the main while-loop one marking is popped from the *Waiting* stack and it is put on the *Passed* list (unless it is already there). Thanks to the first condition at line 30 such a marking will never be pushed again to the *Waiting* stack. This implies the termination of the main while-loop of the algorithm as the other while-loop at lines 20 to 24 surely terminates, latest once the *Trace* stack gets empty. □

In order to prove soundness and completeness of the algorithm, we first highlight an invariant that shows the relationship between the stacks *Waiting* and *Trace*. It is illustrated in Figure 5. The idea is that the liveness algorithm performs a standard depth-first search using the *Waiting* stack and exploring only canonical markings that satisfy the proposition φ and have no more than k tokens. The trace stack stores a path from the initial marking M_0 to the currently explored marking. This fact is formulated by the following lemma.

Lemma 4 (Invariant). *The outer while-loop in Algorithm 1 satisfies the following invariant. If Trace* $= (M'_0, M'_1, \ldots, M'_n)$ *then*

- *Waiting* $= \big(\underbrace{M_0^0, M_0^1, \ldots, M_0^{k_0}}_{k_0 = M_0.successors - 1}, \underbrace{M_1^0, M_1^1, \ldots, M_1^{k_1}}_{k_1 = M_1.successors - 1}, \ldots, \underbrace{M_n^0, M_n^1, \ldots, M_n^{k_n}}_{k_n = M_n.successors \geq 1}\big)$
  where M_i^j is a canonical marking of the j'th so far unexplored M'_i-successor that satisfies φ, and
- *there is a real computation of the net $M_0 \to M_1 \to \cdots \to M_n$ such that $M'_i = cut(M_i)$ for $1 \leq i \leq n$.*

Proof. Both claims of the invariant are trivially satisfied the first time the while-loop is entered. Let us assume the invariant holds before executing the body of the while-loop and let us pop a marking M from the *Waiting* stack at line 8.

- If M is not on the passed list, then all possible firing-successors and 1-delay-successors are generated at lines 12 to 14 and added to the passed/waiting data structure by the call to AddtoPW. Now all canonical successors that are not already on the passed list, satisfy φ and their size is no more than k are pushed on the *Waiting* stack (line 31) and the successor count for the marking M is increased accordingly (line 32).
- If M is already on the passed list, there is nothing more to explore and the successor count for the marking on the top of the *Trace* stack is decreased by one.

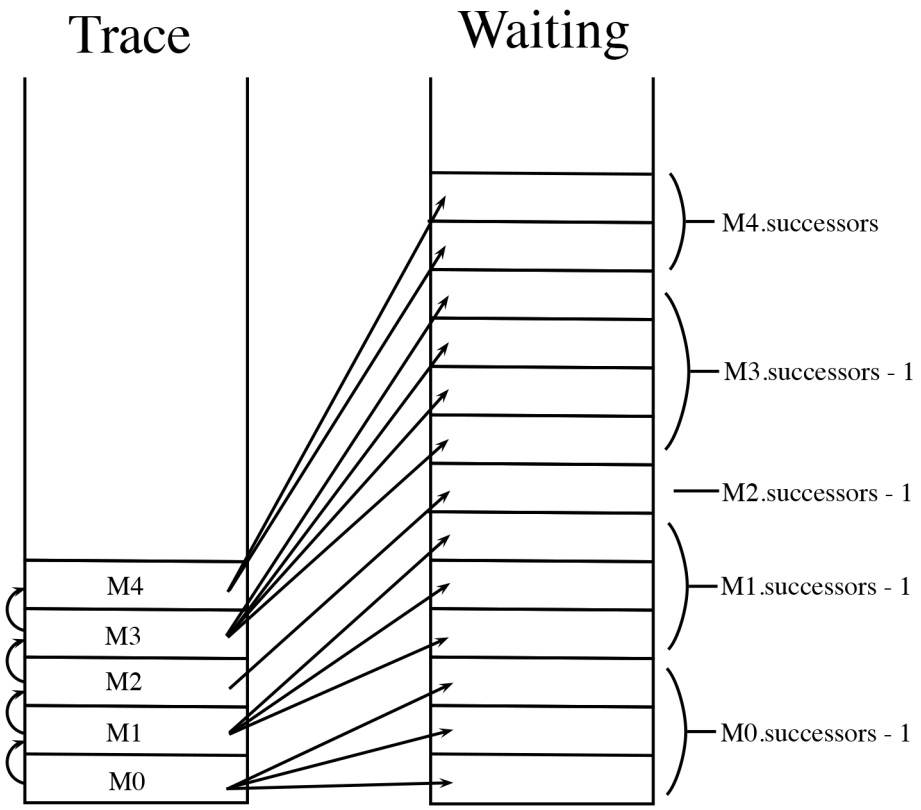

Fig. 5. Invariant property connecting the stacks *Trace* and *Waiting*; arrows point from a marking on the *Trace* stack to its so far unexplored successors that satisfy φ

The inner while-loop at lines 20 to 24 now takes care that during the backtracking all markings on the *Trace* list that have no further successors on the *Waiting* list to explore are popped so that the first part of the invariant is recovered and the top of the *Trace* stack corresponds to a marking that has at least one unexplored successor on the *Waiting* stack.

The second part of the invariant follows from the fact that a canonical marking is pushed to the *Trace* stack at line 11 only if it is a so far unexplored canonical successor of M'_n that is on the top of the *Trace* stack; this is implied by the first part of the invariant. The fact that there is a real execution of the net N going through markings that are equivalent to the canonical ones present in the *Trace* stack follows from Theorem 1. By Lemma 1 part 2 and the fact that $cut(cut(M)) = cut(M)$ we get the claimed property. □

We can now establish the soundness of the algorithm.

Lemma 5 (Soundness). *Let (N, M_0) be a marked TAPN and let $\varphi \in \Phi$ be a proposition. If Algorithm 1 returns true then there is a maximal run $\{M_i\}$ such that $M_i \models \varphi$.*

Proof. The algorithm can return true at two places. At line 17 and at line 29.

In the first case (termination at line 17), we know that the canonical marking M on the top of the *Trace* stack has no successors. By the second part of Lemma 4, there is a real execution of the net $M_0 \to M_1 \to \cdots \to M_n \to \overline{M}$ where by Lemma 1 part 3 all markings satisfy φ and by Theorem 1 and the fact that $M \equiv \overline{M}$ it is a maximal run in the net N as requested.

In the second case (termination at line 29) we know that we found a loop invariantly satisfying φ and consisting of canonical markings. By the same arguments as in the first part of this lemma, there is a real execution that will at the end reach a marking that is equivalent to some other marking already seen during the trace. By repeatedly applying Theorem 1 we can unfold it to an infinite run (not necessarily forming a loop) consisting of markings that satisfy φ. Hence we found a maximal run where φ is invariantly true as requested. □

Finally, we prove the completeness of the liveness algorithm.

Lemma 6 (Completeness). *Let (N, M_0) be a marked TAPN and let $\varphi \in \Phi$ be a proposition. If there is a maximal run $\{M_i\}$ such that $M_i \models \varphi$ and $size(cut(M_i)) \leq k$ for all i then Algorithm 1 returns true.*

Proof. We assume that there is such a maximal run $\{M_i\}$. We have two cases.

- Let $\{M_i\}_{i=0}^{n}$ be a finite maximal run where $M_n \not\rightharpoonup$. Because our algorithm performs a standard depth-first search and there is a trace from M_0 to M_n passing through markings that have a canonical size no more than k and all satisfy φ, we will eventually (unless the algorithm already returned true for some other reason) pop the marking $cut(M_n)$ from the *Waiting* stack and return true at line 17 as $cut(M_n)$ does not enable any transition firing nor allows for a positive time delay.

– Let $\{M_i\}$ is an infinite maximal run. Consider now the sequence of markings $\{cut(M_i)\}$ that are explored in our search algorithm. As the size of each canonical marking in this infinite sequence is bounded by k and the ages of the tokens in each place p are bounded by $C_{max}(p)+1$, there are only finitely many such canonical markings. This means that there are some indices j and k, $0 \leq j < k$, such that $cut(M_j) = cut(M_k)$. In our search algorithm, unless it already returned true, we will eventually get to the situation when the first marking $M \in \{M_j, M_{j+1}, \ldots, M_k\}$ is popped from the *Waiting* stack at line 8. This means that none of the markings M_j, M_{j+1}, …, M_k forming the loop is on the *Passed* list yet. Hence the depth-first search from M will eventually discover again the same marking M and at line 28 the algorithm detects this and returns true as claimed by the lemma.

□

D Verification of Lynch-Shavit Protocol

The TAPN model of the protocol was taken from [1] and we check for the existence of a possible schedule (EG query) that guarantees that the critical section is repeatedly entered within a given interval. The details about the TAPAAL model of the protocol can be found in [8] and the model with the liveness query is available from the download section at http://www.tapaal.net. The verification times (in seconds) for the native UPPAAL model, the translations and our algorithm are presented in Table 3. The protocol documents a similar performance as in case of Fischer's protocol, showing even more clearly that by scaling the size of the problem (number of processes), the size of constants where we can verify the problem faster than the DBM-based methods is growing more then linearly.

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	1.0	2.5	5.4	11.0
6	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	1.2	1.2	1.2	1.1	1.2	1.1	1.3
	0.1	0.2	0.8	2.6	7.5	18.7	40.7
7	5.1	5.1	5.1	5.1	5.1	5.1	5.1
	51.3	51.2	51.4	51.2	51.3	52.5	51.5
	0.1	**0.4**	**1.7**	6.5	20.8	58.4	139.5
8	456.2	457.0	450.1	450.6	458.6	459.4	452.7
	◷	◷	◷	◷	◷	◷	◷
	0.1	**0.6**	**3.4**	**15.0**	**53.2**	**159.2**	**425.9**
9	◷	◷	◷	◷	◷	◷	◷
	◷	◷	◷	◷	◷	◷	◷
	0.1	**1.0**	**6.5**	**31.7**	**125.4**	**412.7**	◷

Table 3. Lynch-Shavit 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 translation using the UPPAAL engine (the fastest one), third line is the discrete TAPAAL engine; ◷ stands for more than 900 seconds