Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Discrete and Continuous Strategies for Timed-Arc Petri Net Games*

Peter Gjøl Jensen, Kim Guldstrand Larsen, Jiří Srba

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

Received: November 2016 / Revised version: May 2017

Abstract. Automatic strategy synthesis for a given control objective can be used to generate correct-by-construction controllers of real-time reactive systems. The existing symbolic approach for continuous timed games is a computationally hard task and current tools like UPPAAL TiGa often scale poorly with the model complexity. We suggest an explicit approach for strategy synthesis in the discrete-time setting and show that even for systems with closed guards, the existence of a safety discrete-time strategy does not imply the existence of a safety continuous-time strategy and vice versa. Nevertheless, we prove that the answers to the existence of discrete-time and continuous-time safety strategies coincide on a practically motivated subclass of urgent controllers that either react immediately after receiving an environmental input or wait with the decision until a next event is triggered by the environment. We then develop an on-the-fly synthesis algorithm for discrete timed-arc Petri net games. The algorithm is implemented in our tool TAPAAL and based on the experimental evidence, we discuss the advantages of our approach compared to the symbolic continuous-time techniques.

1 Introduction

Formal methods and model checking techniques have traditionally been used to verify whether a given system model complies with its specification. However, when we consider formal (game) models where both the controller and the environment can make choices, the question now changes to finding a controller strategy such that any behaviour under such a fixed strategy complies with the given specification. The model checking approach can be used as a try-and-fail technique to check whether a given controller is correct but automatic synthesis of a correct-by-construction controller, as already proposed by Church [12,11], is a more difficult problem as documented e.g. by the SYNTCOMP competition and SYNT workshop [23]. The area has recently seen renewed interest, partly given the rise in computational power that makes synthesis feasible. We focus on the family of timed systems, where for the model of timed automata [1] synthesis has already been proposed [35] and implemented [3,10].

In the area of model checking, symbolic continuous-time on-the-fly methods were ensuring the success of tools such as Kronos [8], UPPAAL [4], Tina [5] and Romeo [20], utilizing the zone abstraction approach [1] via the data structure DBM [16]. These symbolic techniques were recently employed in on-the-fly algorithms [30] for synthesis of controllers for timed games [3, 10,35]. While these methods scale well for classical reachability, the limitation of symbolic techniques is more apparent when used for liveness properties and for solving timed games. We have shown that for reachability and liveness properties, the discrete-time methods performing point-wise exploration of the state-space can prove competitive on a wide range of problems [2], in particular in combination with additional techniques as time-darts [27], constant-reducing approximation techniques [6] and memory-preserving data structures like PTrie [26,25].

In this paper, we benefit from the recent advances in the discrete-time verification of timed systems and suggest an on-the-fly point-wise algorithm for the synthesis of timed controllers relative to safety objectives (avoiding undesirable behaviour). The algorithm is described for a novel game extension of the well-studied timed-arc Petri net formalism [7,21] and we show that in the general setting, the existence of a controller for a

* Extended version of [24] with full proofs, improved implementation and updated experimental results.

safety objective in the discrete-time setting does not imply the existence of such a controller in the continuous-time setting and vice versa, not even for systems with closed guards—contrary to the fact that continuous-time and discrete-time reachability problems coincide for such timed models [9], in particular also for timed-arc Petri nets [32]. However, if we restrict ourselves to the practically relevant subclass of urgent controllers that either react immediately to the environmental events or simply wait for another occurrence of such an event, then we can use the discrete-time methods for checking the existence of a continuous-time safety controller on closed timed-arc Petri nets. The algorithm for controller synthesis is implemented in our open source tool TAPAAL [15], including the memory optimization technique via PTrie [26,25]. The experimental data show a promising performance on a large data-set of infinite job scheduling problems and scaled instances of the disk operation scheduling problem.

Related Work. An on-the-fly algorithm for synthesizing continuous-time controllers for both safety, reachability and time-optimal reachability for time automata was proposed by Cassez et al. [10] and later implemented in the tool UPPAAL TiGa [3]. This work is based on the symbolic verification techniques invented by Alur and Dill [1] in combination with ideas on synthesis by Pnueli et. al [35] and on-the-fly dependency graph algorithms suggested by Liu and Smolka [30]. For timed games, abstraction refinement approaches have been proposed and implemented by Peter et al. [33,34] and Finkbeiner et al. [19] as an attempt to speed up synthesis, while using the same underlying symbolic representation as UPPAAL TiGa. These abstraction refinement methods are complementary to the work presented here. Our work uses the formalism of timed-arc Petri nets that has not been studied in this context before and we rely on the methods with discrete interpretation of time as presented by Andersen et. al [2]. As an additional contribution, we implement our solution in the tool TAPAAL, utilizing memory reduction techniques by Jensen et. al [26], and compare the performance of both discrete-time and continuous-time techniques. Control synthesis and supervisory control was also studied for the family of Petri net models [17,18,36,38] but these works do not consider the timing aspects.

2 Disk Operation Scheduling Example

We shall now provide an intuitive description of the timed-arc Petri net game of *disk operation scheduling* in Figure 1, modelling the scheduler of a mechanical hard-disk drive (left) and a number of read stream requests (right) that should be fulfilled within a given deadline D. The net consists of *places* drawn as circles (the dashed circle around the places R_1, R_2, R_3 and *Buffer* simply means that these places are shared between the two subnets) and *transitions* drawn as rectangles that are either filled (controllable transitions) or framed only (environmental transitions). Places can contain *tokens* (like the places R_1 to R_3 and the place $track_1$) and each token carries its own age. Initially all token ages are 0. The net also contains *arcs* from places to transitions (input arcs) or transitions to places (output arcs). The input arcs are further decorated with *time intervals* restricting the ages of tokens that can be consumed along the arc. If the time interval is missing, we assume the default $[0, \infty]$ interval not restricting the ages of tokens in any way.

In the initial *marking* (token configuration) depicted in our example, the two transitions connected by input arcs to the place $track_1$ are *enabled* and the controller can decide to *fire* either of them. As the transitions contain a white circle, they are *urgent*, meaning that time cannot pass as long at least one urgent transition is enabled. Suppose now that the controller decides to fire the transition on the left of the place $track_1$. As a result of firing the transition, the two tokens in R_1 and $track_1$ will be consumed and a new token of age 0 is produced to the place W_1. Tokens can be also transported via a pair of an input and output *transport arcs* (not depicted in our example) that will transport the token from the input to the output place while preserving its age.

In the new marking we just achieved, no transition is enabled due to the time interval $[1, 4]$ on the input arc of the environmental transition connected to the place W_1. However, after one time unit passes and the token in W_1 becomes of age 1, the transition becomes enabled and the environment may decide to fire it. On the other hand, the place W_1 also contains the *age invariant* ≤ 4, requiring that the age of any token in that place may not exceed 4. Hence after age of the token reaches 4, time cannot progress anymore and the environment is forced to fire the transition, producing two fresh tokens into the places *Buffer* and $track_1$. Hence, reading the data from track 1 of the disk takes between 1ms to 4ms (depending on the actual rotation of the disk) and it is the environment that decides the actual duration of the reading operation.

The idea is that the disk has three tracks (positions of the reading head) and at each track $track_i$ the controller has the choice of either reading the data from the given track (assuming there is a reading request represented by a token in the place R_i) or move the head to one of the neighbouring tracks (such a mechanical move takes between 1ms to 2ms). The reading requests are produced by the subnet on the right where the environment decides when to generate a reading request in the interval between 6ms to 10ms. The number of tokens in the right subnet represents the parallel reading streams. The net also contains *inhibitor arcs* with a cirle-headed tip that prohibit the environmental transitions from generating a reading request on a given track if there is already one.

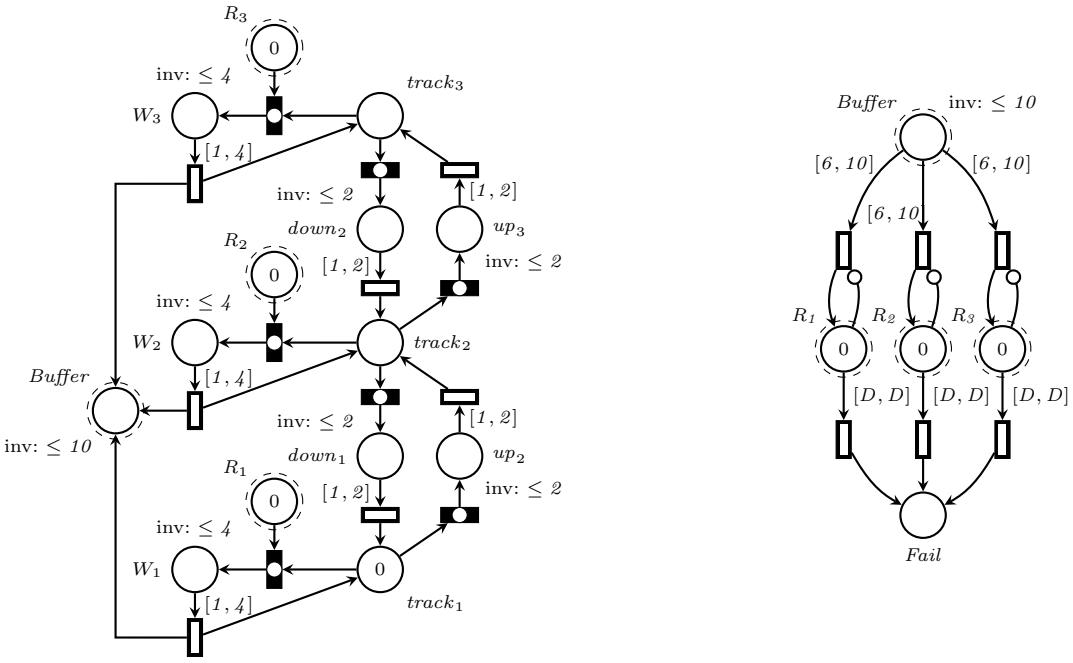


Fig. 1: A timed-arc Petri net game model of a harddisk

Finally, if the reading request takes too long and the age of the token in R_i reaches the age D, the environment has the option to place a token in the place *Fail*.

The control synthesis problem asks to find a strategy for firing the controllable transitions that guarantees no failure, meaning that irrelevant of the behaviour of the environment, the place *Fail* never becomes marked (safety control objective). The existence of such a control strategy depends on the chosen value of D and the complexity of the controller synthesis problem can be scaled by adding further tracks (in the subnet of the left) or allowing for more parallel reading streams (in the subnet on the right). In what follows, we shall describe how to automatically decide in the discrete-time setting (where time can be increased only by nonnegative integer values) whether a controller strategy exists. As the controllable transitions are urgent in our example, the existence of such a discrete-time control strategy implies also the existence of a continuous-time control strategy where the environment is free to fire transitions after an arbitrary delay taken from the dense time domain—a result we formally state and prove in Section 4.

3 Definitions

Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}_0^\infty = \mathbb{N}_0 \cup \{\infty\}$. Let $\mathbb{R}^{\geq 0}$ be the set of all nonnegative real numbers. A *timed transition system* (TTS) is a triple $(S, Act, \to)$ where S is the set of states, Act is the set of actions and $\to \subseteq S \times (Act \cup \mathbb{R}^{\geq 0}) \times S$ is the transition relation written as $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \to$. If $a \in Act$ then we call it a *switch transition*, if $a \in \mathbb{R}^{\geq 0}$ we call it a *delay transition*. We also define the set of *well-formed closed 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 its subset $\mathcal{I}^{\text{inv}} \stackrel{\text{def}}{=} \{[0, b] \mid b \in \mathbb{N}_0^\infty\}$ used in age invariants.

Definition 1 (Timed-Arc Petri Net).
A *timed-arc Petri net* (TAPN) is a 9-tuple $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ where

- P is a finite set of *places*,
- T is a finite set of *transitions* such that $P \cap T = \emptyset$,
- $T_{urg} \subseteq T$ is the set of *urgent transitions*,
- $IA \subseteq P \times T$ 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 such that
  - if $(p, t) \in IA$ and $t \in T_{urg}$ then $g((p, t)) = [0, \infty]$,
- $w : IA \cup OA \to \mathbb{N}$ is a function assigning *weights* to input and output arcs,
- $Type : IA \cup OA \to \mathbf{Types}$ is a *type function* assigning a type to all arcs where $\mathbf{Types} = \{Normal, Inhib\} \cup \{Transport_j \mid j \in \mathbb{N}\}$ such that
  - if $Type(z) = Inhib$ then $z \in IA$ and $g(z) = [0, \infty]$,
  - if $Type((p, t)) = Transport_j$ for some $(p, t) \in IA$ then there is exactly one $(t, p') \in OA$ such that $Type((t, p')) = Transport_j$,
  - if $Type((t, p')) = Transport_j$ for some $(t, p') \in OA$ then there is exactly one $(p, t) \in IA$ such that $Type((p, t)) = Transport_j$,
  - if $Type((p, t)) = Transport_j = Type((t, p'))$ then $w((p, t)) = w((t, p'))$,

– $I : P \to \mathcal{I}^{inv}$ is a function assigning *age invariants* to places.

Remark 1. Note that for transport arcs we assume that they come in pairs (for each type $Transport_j$) and that their weights match. Also for inhibitor arcs and for input arcs to urgent transitions, we require that the guards are $[0, \infty]$. This restriction is important for some of the results presented in this paper and it also guarantees that we can use DBM-based algorithms in the tool TAPAAL [15].

Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPN. We denote by ${}^\bullet x \stackrel{\text{def}}{=} \{y \in P \cup T \mid (y, x) \in IA \cup OA,\ Type((y, x)) \neq Inhib\}$ the preset of a transition or a place x. Similarly, the postset is defined as $x^\bullet \stackrel{\text{def}}{=} \{y \in P \cup T \mid (x, y) \in (IA \cup OA)\}$. Let $\mathcal{B}(\mathbb{R}^{\geq 0})$ be the set of all finite multisets over $\mathbb{R}^{\geq 0}$. A *marking* M on N is a function $M : P \longrightarrow \mathcal{B}(\mathbb{R}^{\geq 0})$ where for every place $p \in P$ and every token $x \in M(p)$ we have $x \in I(p)$, in other words all tokens have to satisfy the age invariants. The set of all markings in a net N is denoted by $\mathcal{M}(N)$.

We write (p, x) to denote a token at a place p with the age $x \in \mathbb{R}^{\geq 0}$. Then $M = \{(p_1, x_1), (p_2, x_2), \ldots, (p_n, x_n)\}$ is a multiset representing a marking M with n tokens of ages x_i in places p_i. We define the size of a marking as $|M| = \sum_{p \in P} |M(p)|$ where $|M(p)|$ is the number of tokens located in the place p.

Definition 2 (Enabledness). Assume a given TAPN $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$. 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.

$$\forall p \in {}^\bullet t.\ x_p^i \in g((p, t)) \text{ for } 1 \leq i \leq w((p, t))$$

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

$$\forall (p, t) \in IA.\, Type((p, t)) = Inhib$$
$$\Rightarrow$$
$$|M(p)| < w((p, t))$$

– 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.

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

– for all normal output arcs, the age of the output token is 0, i.e.

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

A TAPN N defines a TTS $T(N) \stackrel{\text{def}}{=} (\mathcal{M}(N), T, \to)$ 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 *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* $d \in \mathbb{R}^{\geq 0}$ is allowed in M if
  – $(x + d) \in I(p)$ for all $p \in P$ and all $x \in M(p)$, and
  – if $M \xrightarrow{t} M'$ for some $t \in T_{urg}$ then $d = 0$.

  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.

Let $\to \stackrel{\text{def}}{=} \bigcup_{t \in T} \xrightarrow{t} \cup \bigcup_{d \in \mathbb{R}^{\geq 0}} \xrightarrow{d}$. By $M \xrightarrow{d,t} M'$ we denote that there is a marking M'' s.t. $M \xrightarrow{d} M'' \xrightarrow{t} M'$.

The semantics defined above in terms of timed transition systems is called the *continuous-time semantics*. If we restrict the possible delay transitions to take values only from nonnegative integers and the markings to be of the form $M : P \longrightarrow \mathcal{B}(\mathbb{N}_0)$, we call it the *discrete-time semantics*.

An example of a TAPN modeling an office fridge can be seen in Figure 2. Initially, the *Fridge* contains two tokens, representing two boxes of yogurt. If a sudden *Hunger* occurs—which happens every 6 to 24 hours—the yogurts are moved to the *Eat* place. As the arcs moving the yogurts are diamond tipped transport-arcs (with weight two), the ingredients retain their age when moved. At the *Eat* place, we can now either put an uneaten yogurt back to the fridge (by firing the middle transition and hence preserving its age) or eat it and replace it with a new product, resetting the age of the (now replaced) yogurt—this occurs when firing the top transition. Notice that both transitions are urgent, denoted by the rectangle with an empty inner circle. This implies that we need to choose immediately whether or not we consume any of the yogurts. When replacing the yogurt, we also place a token in the *Watching* place. As the new yogurt is precious to us, we will for the next 12 hours be watching the fridge, inhibiting anyone to steal the yogurt—here modeled by an circle-tipped inhibitor arc. After exactly 12 hours, the token in *Watching* is forced to disappear as a combination of the guards *[12,12]* and the invariant $[0, 12]$ abbreviated as inv: ≤ 12. If any yogurt reaches an age between 36 to 42 hours, and we are

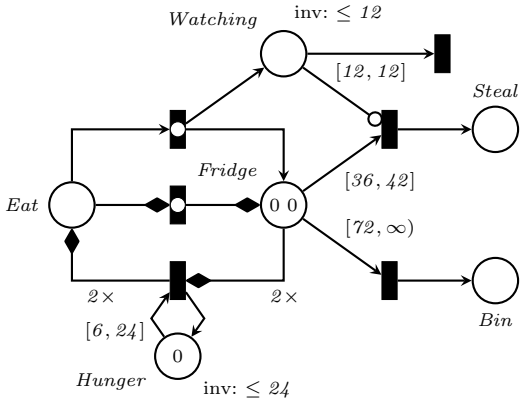


Fig. 2: A TAPN model of the office fridge

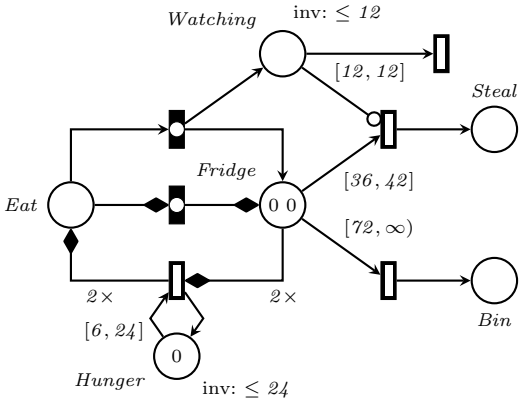


Fig. 3: A TAPG model of the office fridge

not *Watching* the fridge, then someone may *Steal* a yogurt. At the same time, if any of the yogurts gets more than three days old, it may be sent to the *Bin*.

As this describes a plain TAPN, it is always nondeterministically decided what happens when, so it is surely possible that a yogurt gets stolen or is placed to the bin. However, there are some transitions that we are clearly in control of and others that are controlled by the environment. This brings us to the definition of a two player game.

3.1 Timed-Arc Petri Net Game

We extend the TAPN model into the game setting by partitioning the set of its transitions into the controllable and uncontrollable ones.

Definition 3 (Timed-Arc Petri Net Game). A Timed-Arc Petri Net Game (TAPG) is a TAPN with its set of transitions T partitioned into the set of transitions T_{ctrl} owned by the controller and the set T_{env} owned by the environment.

Let us transform our fridge model from Figure 2 into a TAPG, depicted in Figure 3. In the game, we are able to define which transitions are controllable (denoted by solid rectangles, like the two urgent transitions in our figure) and which are not controllable and its firing is determined by the environment (denoted by frame rectangles). Hence we are not in control of when we get hungry or whether other person will steal or throw out our food.

Let G be a fixed TAPG. Recall that $\mathcal{M}(G)$ is the set of all markings over the net G. A *controller strategy* for the game G is a function

$$\sigma : \mathcal{M}(G) \to \mathcal{M}(G) \cup \{wait\}$$

from markings to markings or the special symbol *wait* such that

- if $\sigma(M) = wait$ then either M can delay forever ($M \xrightarrow{d}$ for all $d \in \mathbb{R}^{\geq 0}$), or there is $d \in \mathbb{R}^{\geq 0}$ where $M \xrightarrow{d} M'$ and for all $d'' \in \mathbb{R}^{\geq 0}$ and all $t \in T_{ctrl}$ we have that whenever $M' \xrightarrow{d''} M''$ then $M'' \overset{t}{\not\to}$, and
- if $\sigma(M) = M'$ then there is a $d \in \mathbb{R}^{\geq 0}$ and there is a $t \in T_{ctrl}$ such that $M \xrightarrow{d,t} M'$.

Intuitively, a controller can in a given marking M either decide to wait indefinitely (assuming that it is not forced by age invariants or urgency to perform some controllable transition) or it can suggest a delay followed by a controllable transition firing. The environment can in the marking M also propose to wait (unless this is not possible due to age invariants or urgency) or suggest a delay followed by firing of an uncontrollable transition. If both the controller and environment propose transition firing, then the one preceding with a shorter delay takes place. In the case where both the controller and the environment propose the same delay followed by a transition firing, then any of these two firings can (nondeterministically) happen.

This intuition is formalized in the notion of *plays* following a fixed controller strategy that summarize all possible executions for any possible environment.

Let $\pi = M_1 M_2 \dots M_n \dots \in \mathcal{M}(G)^\omega$ be an arbitrary finite or infinite sequence of markings over G and let M be a marking. We define the concatenation of M with π as $M \circ \pi = M M_1 \dots M_n \dots$ and extend it to the sets of sequences $\Pi \subseteq \mathcal{M}(G)^\omega$ so that $M \circ \Pi = \{M \circ \pi \mid \pi \in \Pi\}$.

Definition 4 (Plays According to the Strategy σ). Let G be a TAPG, M a marking on G and σ a controller strategy for G. We define a function $\mathbb{P}_\sigma : \mathcal{M}(G) \to 2^{\mathcal{M}(G)^\omega}$ returning for a given marking M the set of all possible plays starting from M under the strategy σ.

- If $\sigma(M) = wait$ then $\mathbb{P}_\sigma(M) = \bigcup\{M \circ \mathbb{P}_\sigma(M') \mid d \in \mathbb{R}^{\geq 0},\ t \in T_{env},\ M \xrightarrow{d,t} M'\} \cup X$ where $X = \{M\}$ if

$M \xrightarrow{d}$ for all $d \in \mathbb{R}^{\geq 0}$, or if there is $d' \in \mathbb{R}^{\geq 0}$ such that $M \xrightarrow{d'} M'$ and $M' \xrightarrow{d''} \not\to$ for any $d'' > 0$ and $M' \xrightarrow{t} \not\to$ for any $t \in T_{env}$, otherwise $X = \emptyset$.
- If $\sigma(M) \neq wait$ then according to the definition of controller strategy we have $M \xrightarrow{d,t} \sigma(M)$ and we define $\mathbb{P}_\sigma(M) = \bigcup\{M \circ \mathbb{P}_\sigma(\sigma(M))\} \cup \bigcup\{M \circ \mathbb{P}_\sigma(M') \mid d' \leq d, t' \in T_{env}, M \xrightarrow{d',t'} M'\}$.

The first case says that the plays from the marking M where the controller wants to wait consist either of the marking M followed by any play from a marking M' that can be reached by the environment from M after some delay and firing a transition from T_{env}, or a finite sequence finishing with the marking M if it is the case that M can delay forever, or we can reach a deadlock where no further delay is possible and no transition can fire.

The second case where the controller suggests a transition firing after some delay, contains M concatenated with all possible plays from $\sigma(M)$ and from $\sigma(M')$ for any M' that can be reached by the environment before or at the same time the controller suggests to perform its move.

We can now define the safety objectives for TAPGs. A safety objective is a Boolean expression over arithmetic predicates which observe the number of tokens in the different places of the net. Let φ be so a Boolean combination of predicates of the form $e \bowtie e$ where $e ::= p \mid n \mid e + e \mid e - e \mid e * e$ and where $p \in P$, $\bowtie \in \{<, \leq, =, \neq, \geq, >\}$ and $n \in \mathbb{N}_0$. The semantics of φ in a marking M is given in the natural way, assuming that p stands for $|M(p)|$ (the number of tokens in the place p). We write $M \models \varphi$ if φ evaluates in the marking M to true. We can now state the safety synthesis problem.

Definition 5 (Safety Synthesis Problem). Given a marked TAPG G with the initial marking M_0 and a safety objective φ, decide if there is a controller strategy σ such that

$$\forall \pi \in \mathbb{P}_\sigma(M_0). \forall M \in \pi. M \models \varphi \ . \tag{1}$$

If Equation (1) holds then we say that σ is a *winning controller strategy* for the objective φ.

As an example, we might want to find a strategy for managing our yogurts in the office, modelled in Figure 3. Formally, we can state our goal of not having our food stolen nor thrown out as the safety objective $Steal = 0 \wedge Bin = 0$. One can verify that this can be achieved by e.g. always consuming both yogurts every time hunger strikes. Another safe controller strategy is to return a yogurt into the fridge as long as it is strictly less than 12 hours old, otherwise to eat it.

As another example in Figure 1, we may wish to synthesize, for a given deadline D, a controller for the safety objective $Fail = 0$, hence yielding a controller that can serve three parallel streams with the maximal latency of D. The synthesis problem for this scenario is considerably more complex and with shall return to this problem in our experiments.

4 Synthesis in Continuous and Discrete Time

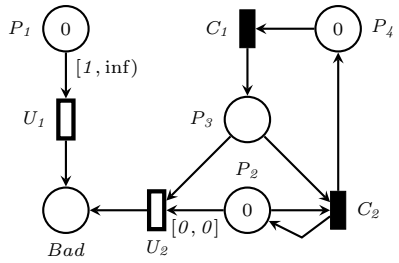
It is known that for classical TAPNs with closed intervals, the continuous and discrete-time semantics coincide up to reachability [32], which is what safety synthesis reduces to if the set of controllable transitions is empty. Contrary to this, Figures 4a and 4b show that this does not hold in general for safety strategies.

For the game in Figure 4a, there exists a strategy for the controller and the safety objective $Bad \leq 0$ but only in continuous-time semantics as the controller has to keep the age of the token in place P_1 strictly below 1, otherwise the environment can mark the place *Bad* by firing U_1. If the controller instead fires transition C_1 without waiting, U_2 becomes enabled and the environment can again break safety. Hence it is impossible to find a discrete-time strategy as even the smallest possible discrete delay of 1 time unit will enable U_1. However, if the controller waits an infinitesimal amount (in the continuous semantics) and fires C_1, then U_2 will not be enabled as the token in P_2 aged slightly. The controller can now fire C_2 and repeat this strategy over and over in order to keep the token in P_1 from ever reaching the age of 1.

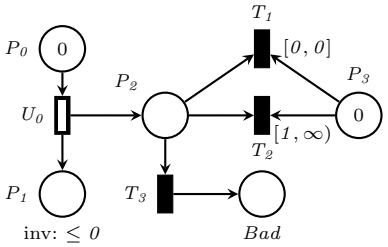
The counter example described before relies on Zeno behaviour, however, this is not needed if we use transport arcs which do not reset the age of tokens (depicted by arrows with diamond-headed tips), as demonstrated in Figure 4c. Here the only strategy for the controller to avoid marking the place *Bad* is to delay some fraction and then fire T_0. Any possible integer delay (1 or 0) will enable the environment to fire U_0 or U_1 before the controller gets to fire T_1. Hence we get the following proposition.

Proposition 1. *There is a TAPG and a safety objective where the controller has a winning strategy in the continuous-time semantics but not in the discrete-time semantics.*

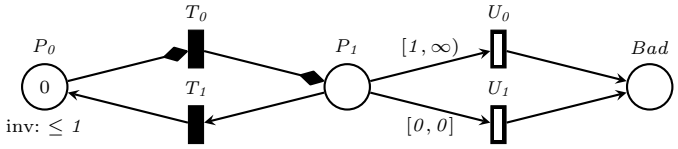
Figure 4b shows, on the other hand, that a safety strategy guaranteeing $Bad \leq 0$ exists only in the discrete-time semantics but not in the continuous-time semantics. Here the environment can mark the place *Bad* by initially delaying 0.5 and then firing U_0. This will produce a token in P_1 which restricts the time from progressing further and thus forces the controller to fire T_3 as this is the only enabled transition. On the other hand, in the discrete-time semantics the environment can either fire U_0 immediately, but then T_1 will be enabled, or it can wait (a minimum of one time unit), which in



(a) A TAPG where $Bad \leq 0$ can be guaranteed by the controller under the continuous-time semantics (by exploiting Zeno behaviour) but not under the discrete-time semantics.



(b) A TAPG where $Bad \leq 0$ can be guaranteed by the controller under the discrete-time semantics but not under the continuous-time semantics.



(c) A TAPG where $Bad \leq 0$ can be guaranteed by the controller under the continuous-time semantics (without exploiting Zeno behaviour) but not under the discrete-time semantics.

Fig. 4: Difference between continuous and discrete-time semantics

turn enables T_2. Hence the controller can in both cases avoid the firing of T_3 in the discrete-time semantics. This implies the following proposition.

Proposition 2. *There is a TAPG and a safety objective where the controller has a winning strategy in the discrete-time semantics but not in the continuous-time semantics.*

This indeed means that the continuous and discrete-time semantics are incomparable and it makes sense to consider both of them, depending on the concrete application domain and whether we consider discretized or continuous time. Nevertheless, there is a practically relevant subclass of the problem where we consider only urgent controllers and where the two semantics coincide. This class contains, for example, all digital circuit-controllers supervising a real-time environment and other applications such as worst-case-optimal controller synthesis for duration probabilistic automata [28].

We say that a given TAPG is with an *urgent controller* if all controllable transitions are urgent, formally $T_{ctrl} \subseteq T_{urg}$. For example the game net in Figure 3 is with urgent controller as the two controllable transitions are both urgent. We can now state our main result of this section, showing that the discrete and continuous semantics coincide for the subclass timed-arc Petri net games with urgent controllers.

Theorem 1. *Let G be a TAPG with urgent controller and let φ be a safety objective. There is a winning controller strategy for G and φ in the discrete-time semantics iff there is a winning controller strategy for G and φ in the continuous-time semantics.*

4.1 Proof of Theorem 1

The rest of this section is now devoted to proving Theorem 1. Clearly, if the urgent controller has a winning strategy while the environment is allowed to make real-time delays, it also has a winning strategy if the environment is only allowed to perform discrete-time delays. We prove the opposite implication by showing that any universal evidence for nonexistence of a controller winning strategy in the continous semantics can be translated into such an evidence in discrete semantics (under the restriction that we consider only urgent controllers). We start by defining a witness for the fact that the controller does not have a winning strategy. A witness can allow for noninteger delays of the environmental moves and the main development of the proof is based on showing that such a witness can be transformed into another one with integer delays only. We do so by translating a witness into a system of linear constraints consisting only of difference constraints which guarantee that if the system has a real solution then it also has an integer solution.

Let us first define a witness for the nonexistence of a controller strategy for a given TAPG G with urgent controller. The intuition of the witness is to provide a strategy for the environment such that it considers all possible choices of the controller. Thus, for the environment choices there are only singleton continuations (if any), and for controller choices there is a multitude of possible continuations.

Definition 6. A *witness* is a function $\gamma : \mathcal{M}(G) \to 2^{\mathcal{M}(G)}$ for a marked TAPG G with urgent controller that for every marking M defines the next possible markings

for the environment to consider. The function γ must satisfy the following conditions for every $M \in \mathcal{M}(G)$.

- If $M \not\models \varphi$ then $\gamma(M) = \emptyset$.
- Else if there is no $d \in \mathbb{R}^{\geq 0}$ and no $t \in T$ such that $M \xrightarrow{d} M' \xrightarrow{t}$ then $\gamma(M) = \emptyset$.
- Else if for all $t \in T_{ctrl}$ it holds that $M \overset{t}{\not\to}$ then $\gamma(M) = \{M'\}$ where for some $d \in \mathbb{R}^{\geq 0}$ and some $t \in T_{env}$ it holds that $M \xrightarrow{d,t} M'$.
- Else
  - either $\gamma(M) = \{M'\}$ such that $M \xrightarrow{t} M'$ for some $t \in T_{env}$, or
  - $\gamma(M) = \{M' \mid M \xrightarrow{t} M', t \in T_{ctrl}\}$ provided that there is at least one $t \in T_{ctrl}$ such that $M \xrightarrow{t}$.

Let us note that as the controller is urgent, the cases above enumerate all next markings to consider once we fix some environmental strategy but still consider all possible controller moves.

Let $\pi \in \mathcal{M}(G)^+$ and by $last(\pi)$ we denote the last marking in the nonempty sequence of markings π. We now define $\Gamma_0 = \{M_0\}$ containing the initial marking and $\Gamma_k = \{\pi \circ M \mid \pi \in \Gamma_{k-1},\ M \in \gamma(last(\pi))\} \cup \{\pi \mid \pi \in \Gamma_{k-1},\ \gamma(last(\pi)) = \emptyset\}$ such that Γ_k describes all possible plays of length at most k under a fixed environmental strategy.

Observe now that a witness γ disproves the existence of any controller winning strategy for the objective φ iff there is $k \in \mathbb{N}$ such that $\Gamma_k = \Gamma_{k+1}$ and for all $\pi \in \Gamma_k$ we have $last(\pi) \not\models \varphi$ showing that every branch of the tree eventually breaks φ. We call such a witness a *counter witness* and denote Γ_k as $\mathbb{P}(\gamma)$. In what follows, whenever the witness function γ for a given marking M returns a set of next markings, we implicitly assume that we know what time delay and transition was fired (according to the definition of γ) in order to reach each individual marking from $\gamma(M)$.

Given a counter witness disproving the existence of any controller winning strategy, we shall now construct a linear constraint system describing a family of witnesses while preserving the plays in $\mathbb{P}(\gamma)$ and alternating only the delays during each play. The goal is to prove that the delays can be altered into integer delays while still preserving the counter witness. The technique of encoding a net computation via a linear constraint system is an adaption of the one used by Mateo et al. [32]. The notation from [32] for describing the linear programs corresponding to a given trace in the system is reused but generalized from a single trace to trees.

Let G be a marked TAPG with urgent controller, and γ be a counter witness for the safety objective φ. A *counter witness tree* is a graph where all plays from $\mathbb{P}(\gamma)$ are organized such that they share prefixes and the edges are labelled by a real-time delay and a transition that was fired after the delay in order to reach the next marking in the play (according to γ). An example of a counter witness tree is given in Figure 5. Here the solid edges correspond to the controller choices (with the implicit delay 0 as we are restricted to an urgent controller) and the dashed edges are the environmental choices where arbitrary real-time delays are possible.

Remark 2. Strictly speaking, organizing the plays so that they share prefixes, may result in the fact that two different plays, after their prefixes diverged, can still both converge later on the same marking M. Hence this situation will not give us a tree structure as the node M will have more than one parent. This is undesirable, so we will implicitly assume that once plays in the game started to differ after a common prefix, any possible markings that are afterwards shared among such plays will appear in the witness tree is several copies (one for each such branch containing a shared marking).

The idea is now to create a constraint system from the counter witness where the concrete delays are replaced by variables, and then solve the resulting constraint system and argue that there is an integer solution to the system. Let us first construct a table Θ, assisting us in creating this constraint system and reflecting the classical firing rule of P/T nets (disregarding the timing constraints for a moment).

The table Θ that serves as a documentation for the counter witness γ and it is given in the form of a matrix with m rows (representing tokens) where m is the maximum number of tokens in any marking in $\mathbb{P}(\gamma)$ and n columns (representing markings in the counter witness tree) where n is the total number of nodes in the counter witness tree. Here we let M and M' to range over specific marking in the counter witness tree (columns) and y range over the tokens in the markings (rows). As each column of the table represents a single marking where not necessarily all m tokens are used, we mark each field $\Theta_{y,M}$ with either $\perp$ (unused token) or the pair (p, f) where $p \in P$ represents the location of the token and $f \in \{0, \bullet\}$ is a flag signalling whether the age of the given token was reset to 0 or left unchanged (represented by the value $\bullet$). We let $\Theta^{place}_{y,M}$ and $\Theta^{flag}_{y,M}$ to denote the elements p and f of the pair of $\Theta_{y,M}$, respectively.

We can now define the notion of a valid table.

Definition 7 (Valid table for a witness γ). Given a counter witness γ with its corresponding witness tree, a table Θ is *valid* if the following conditions are satisfied.

a) For the initial marking $M_0 = \{(p_1, x_1), (p_2, x_2), \ldots, (p_k, x_k)\}$, it holds that the first column of Θ with index M_0 is given by $\Theta_{y,M_0} \overset{\text{def}}{=} (p_y, x_y)$ if $1 \leq y \leq k$, and $\Theta_{y,M_0} \overset{\text{def}}{=} \perp$ if $k < y \leq m$.

b) For each column M in the table where $M \models \varphi$ and an edge $M \xrightarrow{d,t} M'$ in the witness tree, there are two sets $Consume$, $Produce \subseteq \{1, 2, \ldots, m\}$ of token indices and a bijection $\mathcal{U} : \{0, \ldots, m\} \to \{0, \ldots, m\}$ such that

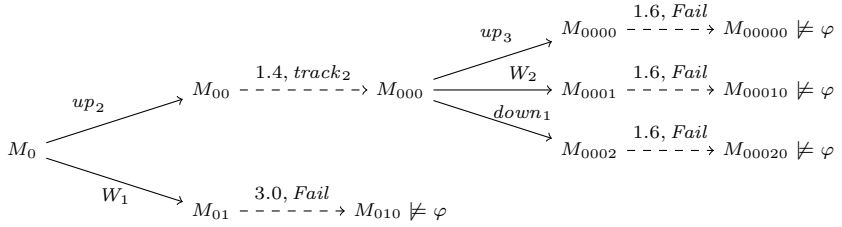


Fig. 5: The tree representation of a counter witness for $\varphi = Fail \leq 0$ for the example in Figure 1 where $D = 3$. The labels of the edges indicate which place will receive a new token by delaying and firing the corresponding transition. Dashed edges indicate the choices of the environment, solid edges are the controller choices (with default delay 0).

- *Consume* is giving the (indices of) tokens consumed when the transition t was fired in the marking M after the delay of d time units in order to reach the marking M' and where it holds that for all $p \in {}^\bullet t$ we have $w(p,t) = |\{y \in Consume \mid \Theta^{place}_{y,M} = p\}|$, and for all $p \in P \setminus {}^\bullet t$ we have $\{y \in Consume \mid \Theta^{place}_{y,M} = p\} = \emptyset$,
- *Produce* is giving the (indices of) tokens produced in the column M' by firing the transition t and where it holds that for all $p \in t^\bullet$ we have $w(t,p) = |\{y \in Produce \mid \Theta^{place}_{y,M'} = p\}|$ and for all $p \in P \setminus t^\bullet$ we have $\{y \in Produce \mid \Theta^{place}_{y,M'} = p\} = \emptyset$, and
- the bijection $\mathcal{U} : \{1, \ldots, m\} \to \{1, \ldots, m\}$ maps the indices of column M to those in the column M' such that
  1. if $|Consume| \leq |Produce|$ and $y \in Consume$ then $\mathcal{U}(y) \in Produce$,
  2. if $|Consume| \geq |Produce|$ and $\mathcal{U}(y) \in Produce$ then $y \in Consume$,
  3. if $y \in Consume$ and $Type((\Theta^{place}_{y,M}, t)) = Transport_j = Type((t,p'))$ then $\mathcal{U}(y) = y$ and $\Theta^{place}_{y,M'} = p'$,
  4. if $y \in \{1, \ldots, m\} \setminus Consume$ and $\Theta_{y,M} \neq \bot$ then $\mathcal{U}(y) = y$ and $\Theta^{place}_{y,M'} = \Theta^{place}_{y,M}$,
  5. if $y \in \{1, \ldots, m\} \setminus Consume$ and $\Theta^{place}_{y,M} = \bot$ then either $\mathcal{U}(y) \in Produce$, or $\mathcal{U}(y) = y$ and $\Theta^{place}_{y,M'} = \bot$, and
  6. if $\Theta_{\mathcal{U}(y),M'} = \bot$ then $y \in Consume$ or $\Theta^{place}_{y,M'} = \bot$.
- and for the column M' in the table holds:
  - if $Type((p,t)) = Inhib$ for some $p \in P$ then $|\{y \in \{1, \ldots, m\} \mid \Theta^{place}_{y,M} = p\}| < w(p,t)$
  - for all $y \in Produce$, if $Type((t, \Theta^{place}_{y,M'})) = Normal$ then $T^{flag}_{y,M'} = 0$ else $T^{flag}_{y,M'} = \bullet$, and
  - if $y \notin Produce$ and $\Theta_{y,M'} \neq \bot$ then $T^{flag}_{y,M'} = \bullet$.

We can now transform the counter witness tree from Figure 5, into such a table, as presented in Table 1.

By following the indices, and the columns given by the table, one can reconstruct an untimed version of the tree given in Figure 5, verifying that the table encodes a valid tree of traces in the classical untimed semantics of Petri nets.

Each column of the table with index M now defines the corresponding untimed marking $u(M)$ as follows.

Definition 8 (Untimed marking for column M). Let M be a column index in a valid table. We define the untimed marking $u(M) \stackrel{\text{def}}{=} \{\Theta^{place}_{y,M} \in P \mid 1 \leq y \leq m\}$ as a multiset of all places where a token is present in the column M of the table.

From the construction, we can verify the validity of the following lemma.

Lemma 1 (Untimed consistency of a valid table). *Let Θ be a valid table and M and M' two of its column indices such that $M \xrightarrow{d,t} M'$. Then $u(M) \xrightarrow{t} u(M')$ in the classical (untimed) Petri net semantics.*

While the construction so far preserves the movement of tokens, it does not encode the restrictions of token ages. We continue by encoding these timing constraints imposed by guards, age invariants and urgency.

We introduce first some notation. Let $M \xrightarrow{d,t} M'$ be an edge in the witness tree. By e_M we denote the global *execution time* of the transition t, yielding the total time elapsed since the computation started from the initial marking until the transition t was fired. Note that if $t \in T_{env}$ then from M there is a unique outgoing edge in the witness tree and if $t \in T_{ctrl}$ then there can be several outgoing edges but all delays on such edges are 0 as we deal with urgent controller. Hence the global execution time of firing t can be associated to the marking M. Now we can define a shorthand $age(y, M)$ for the age of the token given by the row y in a valid table for γ, just at the moment of firing the transition t.

Definition 9 (Token-age expression). Let Θ be a valid table for γ and let M be its column index. We define $age(y, M)$, where $1 \leq y \leq m$, as the expression

$$\text{“ } e_M - e_{M_{j-1}} \text{ ”}$$

such that in the witness tree for γ with the branch $M_0 M_1 \ldots M_{j-1} M_j M_{j+1} \ldots M_i \ldots M_k \in \mathbb{P}(\gamma)$ where

	M_0	M_{00}	M_{000}	M_{0000}	M_{00000}	M_{0001}	M_{00010}	M_{0002}	M_{00020}	M_{01}	M_{010}
1	$(R_1, 0)$	$(R_1, \bullet)$	$(R_1, \bullet)$	$(R_1, \bullet)$	$(Fail, 0)$	$(R_1, \bullet)$	$(Fail, 0)$	$(R_1, \bullet)$	$(Fail, 0)$	$(W_1, 0)$	$(W_1, \bullet)$
2	$(R_2, 0)$	$(R_2, \bullet)$	$(R_2, \bullet)$	$(R_2, \bullet)$	$(R_2, \bullet)$	$(W_2, 0)$	$(W_2, \bullet)$	$(R_2, \bullet)$	$(R_2, \bullet)$	$(R_2, \bullet)$	$(Fail, 0)$
3	$(R_3, 0)$	$(R_3, \bullet)$	$(R_3, \bullet)$	$(R_3, \bullet)$	$(R_3, \bullet)$	$(R_3, \bullet)$	$(R_3, \bullet)$	$(R_3, \bullet)$	$(R_3, \bullet)$	$(R_3, \bullet)$	$(R_3, \bullet)$
4	$(track_1, 0)$	$(up_2, 0)$	$(track_2, 0)$	$(up_3, 0)$	$(up_3, \bullet)$	$\bot$	$\bot$	$(down_1, 0)$	$(down_1, \bullet)$	$\bot$	$\bot$

Table 1: A valid table for the counter witness tree in Figure 5. The rows track the placement of the four tokens we have in the game (here $\bot$ means that a token is not present in the net). Otherwise each cell indicates where is the token located (first coordinate) and whether its age did not change compared to the previous marking (the value $\bullet$ in the second component) or if it was reset to the age 0 (again indicated in the second component).

$M = M_i$ and it is the case that $\Theta^{flag}_{y,M_j} = 0$ and $\Theta^{flag}_{y,M_{j+1}} = \Theta^{flag}_{y,M_{j+2}} = \ldots = \Theta^{flag}_{y,M_i} = \bullet$. By convention M_{-1} is replaced with 0, such that i.e. $age(y, M_0) = e_{M_0}$.

We can now construct a system of inequalities from a valid table Θ.

Definition 10 (Constraint system). Let Θ be a valid table for a counter witness γ. The constraint system $\mathcal{C}$ for Θ is the set of inequations over the variables e_M where M is a column index of Θ and $\mathcal{C}$ is constructed as follows. For each two column indices M and M' with an edge $M \xrightarrow{d,t} M'$ in the witness tree for γ, we

- add to $\mathcal{C}$ the constraint $e_M \leq e_{M'}$, and
- if $\Theta^{place}_{y,M} = p$ and $y \in Consume$[1] and $(p, t) \in IA$ and $g((p, t)) = [\ell, u]$, we add $\ell \leq age(y, M)$ and if $u \neq \infty$ also $age(y, M) \leq u$ to $\mathcal{C}$, and
- if M' enables some urgent transition then we add $e_{M'} - e_M = 0$ to $\mathcal{C}$.

If the initial marking enables some urgent transition then we also add the constraint $e_{M_0} = 0$. Finally, we add the inequalities for age invariants such that for all $y \in \{1, \ldots, m\}$ and all column indices M:

- if $\Theta^{place}_{y,M} = p$ and $I(p) = [0, u]$ where $u \in \mathbb{N}_0$, we add the inequality $age(y, M) \leq u$ to $\mathcal{C}$.

Given our witness from Figure 5, translated into a valid table in Table 1, we can now construct the constraint system as follows. In order to simplify the notation, we shall write e.g. e_{010} instead of $e_{M_{010}}$.

- First, we add the inequalities for preserving the ordering of transition in the witness tree:

$$e_0 \leq e_{00},\ e_{00} \leq e_{000},\ e_{000} \leq e_{0000},\ e_{0000} \leq e_{00000}$$

$$e_{000} \leq e_{0001},\ e_{0001} \leq e_{00010}$$

$$e_{000} \leq e_{0002},\ e_{0002} \leq e_{00020}$$

$$e_0 \leq e_{01},\ e_{01} \leq e_{010}\ .$$

- For the nontrivial age invariants, we add

$$age(2, M_{0001}) \leq 4,\ age(2, M_{00010}) \leq 4$$
$$age(1, M_{01}) \leq 4,\ age(1, M_{010}) \leq 4$$
$$age(4, M_{00}) \leq 2,\ age(4, M_{0000}) \leq 2$$
$$age(4, M_{00000}) \leq 2,\ age(4, M_{0002}) \leq 2$$
$$age(4, M_{00020}) \leq 2$$

  that expand to

$$e_{0001} - e_{000} \leq 4,\ e_{00010} - e_{000} \leq 4$$
$$e_{01} - e_0 \leq 4,\ e_{010} - e_0 \leq 4$$
$$e_{00} - e_0 \leq 2,\ e_{0000} - e_{000} \leq 2$$
$$e_{00000} - e_{000} \leq 2,\ e_{0002} - e_{000} \leq 2$$
$$e_{00020} - e_{0002} \leq 2\ .$$

- For urgency, we add the constraints

$$e_0 = 0,\ e_{000} - e_{00} = 0\ .$$

- Finally for the guards on input arcs, we add the constraints

$$1 \leq age(4, M_{00}) \leq 2,\ 3 \leq age(1, M_{0000}) \leq 3$$
$$3 \leq age(1, M_{0001}) \leq 3,\ 3 \leq age(1, M_{0002}) \leq 3$$
$$3 \leq age(2, M_{01}) \leq 3$$

  that expand to

$$1 \leq e_{00} - e_0 \leq 2,\ 3 \leq e_{0000} \leq 3$$
$$3 \leq e_{0001} \leq 3,\ 3 \leq e_{0002} \leq 3$$
$$3 \leq e_{01} \leq 3\ .$$

One can verify that the original global real-time delays from Figure 5 form a solution of the constructed constraint system: $e_0 = 0$, $e_{00} = 1.4$, $e_{000} = 1.4$, $e_{0000} = 3.0$, $e_{00000} = 3.0$, $e_{0001} = 3.0$, $e_{00010} = 3.0$, $e_{0002} = 3.0$, $e_{00020} = 3.0$, $e_{01} = 3.0$, $e_{010} = 3.0$ Moreover, the constructed equation system also has an integer solution (this is not only a coincidence), e.g. $e_0 = 0$, $e_{00} = 2$, $e_{000} = 2$, $e_{0000} = 3$, $e_{00000} = 3$, $e_{0001} = 3$, $e_{00010} = 3$, $e_{0002} = 3$, $e_{00020} = 3$, $e_{01} = 3$, $e_{010} = 3$ and such a solution also forms a counter witness in our TAPG with urgent controller.

[1] The set *Consume* for the edge $M \xrightarrow{d,t} M'$ was fixed in Definition 7, part b).

Lemma 2. *Let γ be a counter witness for a TAPG G and the safety objective φ.*

a) There is a valid table Θ for γ and the corresponding constraint system $\mathcal{C}$ has a solution.

b) Let e_M where M ranges over the columns of Θ be another solution of $\mathcal{C}$. Then for any play $M_0 M_1 M_2 \dots M_k \in \mathbb{P}(\gamma)$ such that $M_0 \overset{d_0,t_0}{\to} M_1 \overset{d_1,t_1}{\to} M_2 \overset{d_2,t_2}{\to} \dots M_k$, the same sequence of transitions $t_0, t_1, t_2, \dots, t_{k-1}$ can be fired from M_0 with the following delays: $M_0 \overset{e_{M_0},t_0}{\to} M'_1 \overset{e_{M_1}-e_{M_0},t_1}{\to} M'_2 \overset{e_{M_2}-e_{M_1},t_2}{\to} \dots M'_k$ such that $M_i \models \varphi$ iff $M'_i \models \varphi$.

c) The constraint system $\mathcal{C}$ has an integer solution.

Proof. a) From the requirements for a valid table, Lemma 1 and the construction of the constraint system, we can by case analysis verify that if we define $e_{M_i} = d_0 + d_1 + \dots + d_i$ in the play $M_0 M_1 M_2 \dots M_i \dots M_k \in \mathbb{P}(\gamma)$ where $M_0 \overset{d_0,t_0}{\to} M_1 \overset{d_1,t_1}{\to} M_2 \overset{d_2,t_2}{\to} \dots M_i \overset{d_i,t_i}{\to} \dots M_k$, then this forms a solution for the system $\mathcal{C}$. Hence by considering the timing given in the counter witness tree, we have a solution for the constraint system.

b) On the other hand, we notice that the system $\mathcal{C}$ contains all the timing constraints that are necessary for successfully executing all the transitions in the counter witness tree and for producing valid plays. Then simply computing the relative delays before a transition firing can be done by subtracting from the global execution time when the transition is fired the global execution time of the transition that we fired right before. Clearly, as both M_i and M'_i where achieved by firing the same sequence of transitions with just different delays, they have the same token distribution and hence $M_i \models \varphi$ iff $M'_i \models \varphi$.

c) As the constraint system $\mathcal{C}$ uses only difference constraints (difference of at most two variables is compared with a constant), it falls within the special subset of linear programming problems with totally unimodular matrices [13]. For this specific subclass, solving the constraint-system reduces to a shortest-path problem with integer weights only. This reduction implies that an integer solution of such a system exists [14,22], provided that the system is solvable, which it is by part a) of the lemma. □

From Lemma 2 we have that if it is possible to construct a counter witness for the existence of a controller strategy in the continuous-time semantics, then we can translate such a counter witness into a counter witness with integer delays only. This concludes the proof of Theorem 1.

5 Discrete-Time Algorithm for Synthesis

We shall now define the discrete-time algorithm for synthesising controller strategies for TAPGs. As the state-space of a TAPG is infinite in several aspects (the number of tokens in reachable markings can be unbounded and even for bounded nets the ages of tokens can be arbitrarily large), the question of deciding the existence of a controller strategy is in general undecidable (already the classical reachability is undecidable [37] for TAPNs).

We address undecidability by fixing a constant k, bounding the number of tokens in any marking reached by the controller strategy. This means that instead of checking the safety objective φ, we verify instead the safety objective $\varphi_k = \varphi \wedge k \geq \sum_{p \in P} p$ that at the same time ensures that the total number of tokens is at most k. This will, together with the extrapolation technique below, guarantee the termination of the algorithm. We note that in case the net is bounded, there is always some constants k for which checking the property φ_k is equivalent to the original safety property φ and hence the analysis is both sound and complete in this case.

5.1 Extrapolation of TAPGs

We shall now recall a few results from [2] that allow us to make finite abstractions of bounded nets (in the discrete-time semantics). The theorems and lemmas in the rest of this section also hold for continuous-time semantics, however, the finiteness of the extrapolated state space is not guaranteed in this case.

Let $G = (P, T, T_{env}, T_{ctrl}, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPG. In [2] the authors provide an algorithm for computing a function $C_{max} : P \to (\mathbb{N}_0 \cup \{-1\})$ returning for each place $p \in P$ the maximum constant associated to this place, meaning that the ages of tokens in place p that are strictly greater than $C_{max}(p)$ are irrelevant. The function $C_{max}(p)$ for a given place p is computed by essentially taking the maximum constant appearing in any outgoing arc from p and in the place invariant of p, where a special care has to be taken for places with outgoing transport arcs (details are discussed in [2]). In particular, places where $C_{max}(p) = -1$ are the so-called *untimed* places where the age of tokens is not relevant at all, implying that all the intervals on their outgoing arcs are $[0, \infty]$.

Let M be a marking of G. We split it into two markings $M_>$ and $M_\leq$ where $M_>(p) = \{x \in M(p) \mid x > C_{max}(p)\}$ and $M_\leq(p) = \{x \in M(p) \mid x \leq C_{max}(p)\}$ for all places $p \in P$. Clearly, $M = M_> \uplus M_\leq$.

We say that two markings M and M' in the net G are equivalent, written $M \equiv M'$, if $M_\leq = M'_\leq$ and for all $p \in P$ we have $|M_>(p)| = |M'_>(p)|$. This means that for tokens with ages below the maximum constants M and M' agree and also on the number of tokens above the maximum constant. Let us here introduce the notion of timed bisimilarity (we refer e.g. to [29] for more information).

Definition 11 (Timed Bisimulation). A binary relation $\mathcal{R}$ over the markings is a timed bisimulation if for any two markings such that $M\mathcal{R}\hat{M}$ we have

– if $M \xrightarrow{d} M'$ then $\hat{M} \xrightarrow{d} \hat{M}'$ such that $M'\mathcal{R}\hat{M}'$,
– if $\hat{M} \xrightarrow{d} \hat{M}'$ then $M \xrightarrow{d} M'$ such that $M'\mathcal{R}\hat{M}'$,
– if $M \xrightarrow{t} M'$ then $\hat{M} \xrightarrow{t} \hat{M}'$ such that $M'\mathcal{R}\hat{M}'$, and
– if $\hat{M} \xrightarrow{t} \hat{M}'$ then $M \xrightarrow{t} M'$ such that $M'\mathcal{R}\hat{M}'$.

This means that delays and transition firings on one side can be matched by exactly the same delays and transition firings on the other side and vice versa.

We can now see that the above defined relation $\equiv$ is an equivalence relation and it is also a timed bisimulation.

Theorem 2 ([2]). *The relation $\equiv$ is a timed bisimulation.*

We can now define a function computing canonical representatives for each equivalence class of $\equiv$.

Definition 12 (Cut). Let M be a marking. We define its canonical marking $cut(M)$ by $cut(M)(p) = M_{\leq}(p) \uplus \{\underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_{>}(p)| \text{ times}}\}$.

The idea of cut is to utilize our knowledge from Theorem 2 that for any two markings which are timed bisimilar, it is sufficient to only do computations on one of them. The *cut* function takes this one step further and defines, for a group of bisimilar markings, a single canonical representative M capable of exhibiting the same behaviour as any marking M' where $M = cut(M')$.

Lemma 3 ([2]). *Let M, M_1 and M_2 be markings. Then (i) $M \equiv cut(M)$, and (ii) $M_1 \equiv M_2$ if and only if $cut(M_1) = cut(M_2)$.*

Note that as our safety objective φ deals only with the number of tokens in places but not with their actual ages, we get that $M \models \varphi$ if and only if $cut(M) \models \varphi$ for any marking M.

5.2 The Algorithm

After having introduced the extrapolation function *cut* and our enforcement of the k-bound, we can now design an algorithm for computing a controller strategy σ, provided such a strategy exists.

Algorithm 1 describes a discrete-time method to check if there is a controller strategy or not. It is centered around four data structures: *Waiting* set for storing markings to be explored, *Losing* set that contains marking where such a strategy does not exist, *Depend* function for maintaining the set of dependencies to be reinserted to the waiting list whenever a marking is declared as losing, and *Processed* set for already processed markings. All markings in the algorithm are always considered modulo the *cut* extrapolation. The algorithm performs a forward search by repeatedly selecting a marking M from *Waiting* and if it can determine that the controller cannot win from this marking, then M gets inserted into the set *Losing* while the dependencies of M are put to the set *Waiting* in order to backward propagate this information. If the initial marking is ever inserted to the set *Losing*, we can terminate and announce that a controller strategy does not exist. If this is not the case and there are no more markings in the set *Waiting*, then we terminate with success. In this case, it is also easy to construct the controller strategy by making choices so that the set *Losing* is avoided.

To prove that our algorithm is correct, we prove termination, soundness, and completeness.

Lemma 4 (Termination). *Algorithm 1 terminates.*

Proof. Let us first argue that the sets *Waiting*, *Losing* and *Processed* can contain only an a priori bounded number of markings. To see this, we observe that markings added to *Processed* and *Losing* are only those that were previously removed from *Waiting*. Therefore it is sufficient to show that the number of different markings in *Waiting* is bounded. From the definition of φ_k at line 2, we can see that a marking satisfies φ_k only if is has at most k tokens and due to the test at line 23, only such markings can be inserted to *Waiting*. For the given number k, we notice that there are only finitely many extrapolated (by the function *cut*) markings with at most k tokens. Hence the set *Waiting* can contain only a bounded number of different extrapolated markings.

Unless the test at line 4 succeeds and we immediately terminate due to the check at line 8, the following invariant will hold during the execution of the main while-loop at lines 8 to 26: $Waiting \cap Losing = \emptyset$. This is due to the fact that at every line where we add markings to *Waiting* (lines 15, 19 and 25), we are guaranteed that such markings are not in the set *Losing*, and we only add a new marking to *Losing* (lines 14 and 18) when the marking was just popped from *Waiting* at line 9. Similarly, it is easy to observe that during the execution of the algorithm, the sets *Losing* and *Processed* are only growing (there are no lines that ever remove elements from these sets).

In each iteration of the while-loop, we can observe that the size of the set *Waiting* is either decreased by popping an element at line 9, or if new elements are added to *Waiting* then either the cardinality of *Losing* or *Processed* increases by one. We show that by analysing the lines where new elements are possibly added to *Waiting*. This can happen at lines 15 and 19 but at the previous lines 14 and 18, respectively, the marking M got inserted into *Losing*. Because this M was just popped from *Waiting* at line 9 and due to the previously introduced fact that $Waiting \cap Losing = \emptyset$, the cardinality of

Algorithm 1: Safety Synthesis Algorithm

Input: A TAPG $G = (P, T, T_{env}, T_{ctrl}, T_{urg}, IA, OA, g, w, Type, I)$, initial marking M_0, a safety objective φ, a bound k.
Output: *tt* if there exists a controller strategy ensuring φ from M_0 and not exceeding k tokens in any intermediate marking, *ff* otherwise

1 **begin**
2 $Waiting := Losing := Processed := \emptyset$; $\varphi_k = \varphi \wedge k \geq \sum_{p \in P} p$;
3 $M \leftarrow cut(M_0)$; $Depend[M] \leftarrow \emptyset$;
4 **if** $M \not\models \varphi_k$ **then**
5 $Losing \leftarrow \{M\}$
6 **else**
7 $Waiting \leftarrow \{M\}$
8 **while** $Waiting \neq \emptyset \wedge cut(M_0) \notin Losing$ **do**
9 $M \leftarrow pop(Waiting)$;
10 $Succs_{env} := \{cut(M') \mid t \in T_{env},\ M \xrightarrow{t} M'\}$;
11 $Succs_{ctrl} := \{cut(M') \mid t \in T_{ctrl},\ M \xrightarrow{t} M'\}$;
12 $Succs_{delay} := \begin{cases} \emptyset & \text{if } M \not\xrightarrow{1} \\ \{cut(M')\} & \text{if } M \xrightarrow{1} M' \end{cases}$
13 **if** $\exists M' \in Succs_{env}$ *s.t.* $M' \not\models \varphi_k \vee M' \in Losing$ **then**
14 $Losing \leftarrow Losing \cup \{M\}$;
15 $Waiting \leftarrow (Waiting \cup Depend[M]) \setminus Losing$;
16 **else**
17 **if** $Succs_{ctrl} \cup Succs_{delay} \neq \emptyset \wedge \forall M' \in Succs_{ctrl} \cup Succs_{delay}.\ M' \not\models \varphi_k \vee M' \in Losing$ **then**
18 $Losing \leftarrow Losing \cup \{M\}$;
19 $Waiting \leftarrow (Waiting \cup Depend[M]) \setminus Losing$;
20 **else**
21 **if** $M \notin Processed$ **then**
22 **foreach** $M' \in (Succs_{ctrl} \cup Succs_{env} \cup Succs_{delay})$ **do**
23 **if** $M' \notin Losing \wedge M' \models \varphi_k$ **then**
24 $Depend[M'] \leftarrow Depend[M'] \cup \{M\}$;
25 $Waiting \leftarrow Waiting \cup \{M'\}$;
26 $Processed \leftarrow Processed \cup \{M\}$;
27 **return** *tt if* $cut(M_0) \notin Losing$, *else ff*

Losing is so increased. Another place where a new marking can be added to *Waiting* is at line 25. However, in this case we know that $M \notin Processed$ due to the test at line 21, and this implies that by executing line 26, the cardinality of the set *Processed* increased.

In summary, in each iteration of the while-loop either the size of *Waiting* decreases and if not then either the cardinality of *Losing* or *Processed* increases. As these sets are a priori bounded, we know that eventually the set *Waiting* becomes empty and the algorithm terminates (unless the algorithm already terminated due to the successful check $cut(M_0) \in Losing$ at line 8). □

Having proved the termination, we now continue by proving soundness.

Lemma 5 (Soundness). *If Algorithm 1 returns ff then there is no controller winning strategy from M_0 for the safety objective $\varphi_k = \varphi \wedge k \geq \sum_{p \in P} p$.*

Proof. We prove the lemma by establishing the invariant claiming that there is no controller winning strategy for any marking ever inserted into the set *Losing*. By observing that the algorithm returns *ff* only if $cut(M_0) \in Losing$ and the fact that by Theorem 2 the marking $cut(M_0)$ is losing if and only if M_0 is losing (note that our safety logic only queries the number of tokens in places but not their actual ages), this will conclude the proof of this lemma.

The invariant clearly holds before the while-loop is entered as *Losing* is empty. Assume now that the set *Losing* contains markings for which the controller has no strategy to satisfy φ_k and that we add a new marking M to the set *Losing*. We want to argue that the controller does not have a winning strategy from the marking M. A new marking M can be added only at lines 14 or 18.

- If the marking M was added to *Losing* at line 14 then surely, by the test at line 13, there is an environmental transition $M \xrightarrow{t} M'$ with $t \in T_{env}$ such that $cut(M') \not\models \varphi_k$ (and hence also $M' \not\models \varphi_k$) or $cut(M') \in Losing$. Hence, clearly the controlled cannot have a winning strategy from M as the environment can force the computation to the marking M' that either breaks the safety formula φ_k or $cut(M')$ belongs to *Losing* that contains only markings from which controller cannot win and as $cut(M')$ is los-

ing for the controller, so is the marking M' by using Theorem 2.
- If the marking M was added to *Losing* at line 18 then the environment can let the controller to act from M, implying that the controller has to either fire some $t \in T_{ctrl}$ such that $M \xrightarrow{t} M'$ or perform a unit delay such that $M \xrightarrow{1} M'$ and in both situations the controller cannot win from M' as in the previous case because either $cut(M') \not\models \varphi_k$ or $cut(M') \in Losing$.

The soundness of the approach is hence introduced. □

Finally, we can present the completeness lemma.

Lemma 6 (Completeness). *If Algorithm 1 returns tt then the controller has a winning strategy from M_0 for the safety objective $\varphi_k = \varphi \wedge k \geq \sum_{p \in P} p$.*

Proof. Assume that Algorithm 1 returns *tt*. We shall define a winning strategy for the controller starting from the initial marking M_0. Note that as we consider here discrete-time semantics, it is enough for each marking M visited during a play to determine whether the controller's strategy should suggest (i) to fire some controllable transition without any delay, (ii) to perform a delay of one time unit, or (iii) to do nothing (allowed only if none of the options (i) and (ii) are possible). Such a strategy can, in a straightforward way, be extended to the controller strategy as defined in Section 3.1 that proposes from a marking M to delay d time units followed by firing of a controllable transition, or to wait for ever. In what follows, we shall define a winning strategy for the controller from a marking M by defining it for the marking $cut(M)$. By Theorem 2 such a strategy from $cut(M)$ is a valid winning strategy also for the marking M.

The intuition behind the controller strategy is to make sure that any play includes only markings M such that $cut(M) \in Processed \setminus Losing$ (in the rest of this proof we consider the sets *Processed* and *Losing* after the termination of the algorithm). By examining that the set *Processed* contains only markings that were previously in *Waiting* and that any marking inserted into *Waiting* at line 25 must satisfy the proposition φ_k because of the check at line 23 (the markings inserted to *Waiting* at lines 15 and 19 were in the set *Waiting* earlier), we can see that all markings in $Processed \setminus Losing$ satisfy the formula φ_k. Hence staying in $Processed \setminus Losing$ is a safe controller strategy.

Let us so assume a marking $M \in Processed \setminus Losing$. We shall now determine whether the controller should propose to fire some controllable transition enabled in M, to delay in M for one time unit, or to do nothing. There are three cases to consider.

- If $M \xrightarrow{1} M'$ such that $cut(M') \in Processed \setminus Losing$ then the controller will propose to delay for one time unit. Clearly, there cannot be any $t \in T_{env}$ with $M \xrightarrow{t} M''$ where $M'' \not\models \varphi_k$ or $cut(M'') \in Losing$ as otherwise this would be detected at line 13 and it would imply that $M \in Losing$ (due to the back-propagation of this fact at line 15). This contradicts our assumption that $M \in Processed \setminus Losing$ and hence the controller's choice to delay one time unit is safe here.
- If $M \xrightarrow{1} M'$ but $cut(M') \notin Processed \setminus Losing$ then as $cut(M')$ was surely processes due to the classical forward search implemented at lines 21 to 25, this can only be possible if $cut(M') \in Losing$. As the fact that a marking is added to *Losing* is back-propagated at lines 15 and 19, and because $M \notin Losing$, we know due to the test at line 17 there is at least one $t \in T_{ctrl}$ such that $M \xrightarrow{t} M''$ such that $M'' \models \varphi_k$ and $cut(M'') \notin Losing$. Should this not be the case, then M would end up in the set *Losing* at line 18. This contradicts our initial assumption that $M \in Processed \setminus Losing$. The controller can in this case propose to fire one of such transitions t discussed above and the resulting marking will belong to $Processed \setminus Losing$. Clearly, as in the previous case, any environmental transition enabled in M must also lead to a marking from $Processed \setminus Losing$.
- If $M \not\xrightarrow{1}$ then as before any environmental transition from M leads to a marking from $Processed \setminus Losing$ and if some controller transition is enabled at M then at least one such transition will bring us, by arguments already given above, to $Processed \setminus Losing$.

We have so defined a controller strategy that will visit only markings from $Processed \setminus Losing$ and hence it is a winning controller strategy for the objective φ_k. □

We are now ready to state the correctness of our algorithm that follows from Lemmas 4, 5 and 6.

Theorem 3 (Correctness). *Algorithm 1 terminates and returns tt if and only if there is a controller strategy for the safety objective $\varphi_k = \varphi \wedge k \geq \sum_{p \in P} p$.*

Clearly, if the input Petri net game is k-bounded (there is no reachable marking with more than k tokens) then a marking satisfies φ_k if and only if it satisfies φ and hence our algorithm decides the existence of the controller winning strategy for the safety objective φ (in the discrete-time semantics). For unbounded nets, the existence of such a controller winning strategy is undecidable (already the reachability problem is undecidable for timed-arc Petri nets under discrete-time semantics [37]) but our algorithm can possibly find a controller winning strategy for the objective φ that visits only markings with a bounded number of tokens even for games on unbounded timed-arc Petri nets. There can though still be other controller winning strategies that may require an unbounded number of tokens in the visited markings. Such strategies will not be discovered by our algorithm.

1 Stream	$D = 133$	$D = 173$	$D = 213$	$D = 253$	$D = 293$	$D = 333$	$D = 373$
Tracks	70	90	110	130	150	170	190
TAPAAL	16.86s	36.84s	69.55s	119.68s	182.60s	271.82s	376.48s
UPPAAL	36.41s	76.63s	193.37s	351.17s	509.46s	1022.83s	1604.04s
2 Streams	$D = 19$	$D = 27$	$D = 35$	$D = 43$	$D = 51$	$D = 59$	$D = 67$
Tracks	6	8	10	12	14	16	18
TAPAAL	1.17s	5.61s	19.13s	49.23s	114.23s	225.38s	426.99s
UPPAAL	19.11s	93.46s	436.15s	1675.85s	3328.66s	◷	◷
3 Streams	$D = 17$	$D = 21$	$D = 25$	$D = 29$	$D = 35$	$D = 39$	$D = 43$
Tracks	3	4	5	6	7	8	9
TAPAAL	1.30s	8.5s	38.16s	129.27s	454.08s	1153.65s	◷
UPPAAL	885.56s	◷	◷	◷	◷	◷	◷

Table 2: Time in seconds to find a controller strategy for the disk operation scheduling for the smallest D where such a strategy exists.

6 Experiments

The discrete-time controller synthesis algorithm was implemented in the tool TAPAAL [15] and we evaluate the performance of the implementation by comparing it to UPPAAL TiGa [3] version 0.18, the state-of-the-art continuous-time model checker for timed games. The experiments were run on AMD Opteron 6376 processor limited to using 19 GB of RAM[2] and with one hour timeout (denoted by ◷).

Compared to our experiments presented at [24], the performance of TAPAAL improved as we now use a more efficient PTrie [26] implementation that is both faster and has a smaller memory foot-print than the one used in [24].

6.1 Disk Operation Scheduling

In the disk operation scheduling model presented in Section 2 we scale the problem by changing the number of tracks and the number of simultaneous read streams. An equivalent model using the timed automata formalism was created for UPPAAL TiGa. We then ask whether a controller exists respecting a fixed deadline D for all requests. For each instance of the problem, we report the computation time for the smallest deadline D such that it is possible to synthesize a controller. Notice that the disk operating scheduling game net has an urgent controller, hence the discrete and continuous-time semantics coincide.

The results in Table 2 show that our algorithm scales considerably better than TiGa (that suffers from the large fragmentation of zone federations) as the number of tracks increases (by which we scale the size of the problem) and it is significantly better when we add more read streams (by which we scale the concurrency and consequently also the number of timed tokens/clocks).

[2] UPPAAL TiGa only exists in a 32 bit version, but for none of the tests the 4GB limit was exceeded for UPPAAL TiGa.

6.2 Infinite Job Shop Scheduling

In our second experiment, infinite job shop scheduling, we consider the duration probabilistic automata [31]. Kempf et al. [28] showed that "non-lazy" schedulers are sufficient to guarantee optimality in this class of automata. Here non-lazy means that the controller only chooses what to schedule at the moment when a running task has just finished (the time of this event is determined by the environment). We here consider a variant of this problem that should guarantee an infinite (cyclic) scheduling in which processes—while competing for resources—must meet their deadlines. The countdown of a process is started when its first task is initiated and the process deadline is met if the process is able to execute its last task within the deadline. After such a completed cycle, the process starts from its initial configuration and the deadline-clock is restarted. The task of the controller is to find a schedule such that all processes always meet their deadline. The problem can be modelled using urgent controller, in which case the discrete and continuous-time semantics coincide.

The problem is scaled by the number of parallel processes, number of tasks in each processes and the size of constants used in guards (except the deadline D that contains a considerably larger constant). For each set of scaling parameters, we generated 100 random instances of the problem and report on the number of cases where the tool answered the synthesis problem (within one hour deadline) and if more than 50 instances were solved, we also compute the median of the running time.

The comparison with UPPAAL TiGa in Table 3 shows a trend similar to the previous experiment. Our algorithm scales nicely as we increase the number of tasks as well as the number of processes. This is due to the fact that the zone fragmentation in TiGa increases with the number of parallel components and more distinct guards. When scaling the size of constants, the performance of the discrete-time method gets worse and eventually UPPAAL TiGa can solve more instances.

2 Processes/7-13 tokens

Max Age	10 Tasks		12 Tasks		14 Tasks		16 Tasks		18 Tasks	
5	(100)	**54s**	(**100**)	**118s**	(**100**)	**238s**	(**100**)	**464s**	(**100**)	**661s**
$D \leq 144$	(100)	100s	(98)	413s	(85)	1201s	(35)	🕒	(18)	🕒
10	(**100**)	270s	(**100**)	**699s**	(**98**)	**1281s**	(**87**)	**2370s**	(**28**)	🕒
$D \leq 288$	(96)	**221s**	(69)	1443s	(43)	🕒	(16)	🕒	(1)	🕒
15	(**100**)	852s	(**85**)	2043s	(28)	🕒	(**15**)	🕒	(**5**)	🕒
$D \leq 432$	(87)	**315s**	(60)	**1960s**	(19)	🕒	(8)	🕒	(0)	🕒
20	(84)	1982s	(23)	🕒	(14)	🕒	(4)	🕒	(**2**)	🕒
$D \leq 576$	(**90**)	**554s**	(**66**)	**2914s**	(**34**)	🕒	(4)	🕒	(1)	🕒

3 Processes/10-19 tokens

Max Age	2 Tasks		3 Tasks		4 Tasks		5 Tasks		6 Tasks	
5	(**100**)	**2s**	(**100**)	**33s**	(**100**)	**295s**	(**71**)	**1375s**	(**42**)	🕒
$D \leq 57$	(99)	16s	(69)	1827s	(4)	🕒	(0)	🕒	(0)	🕒
10	(**100**)	**14s**	(**99**)	**328s**	(**50**)	**3538s**	(**20**)	🕒	(**8**)	🕒
$D \leq 114$	(98)	32s	(52)	3338s	(6)	🕒	(0)	🕒	(0)	🕒
15	(**100**)	44s	(**73**)	**1052s**	(**32**)	🕒	(**6**)	🕒	(**1**)	🕒
$D \leq 171$	(98)	**27s**	(50)	🕒	(1)	🕒	(0)	🕒	(0)	🕒

4 Processes/13-25 tokens

Max Age	2 Tasks		3 Tasks		4 Tasks		5 Tasks		6 Tasks	
5	(**95**)	**178s**	(**35**)	🕒	(**9**)	🕒	(**1**)	🕒	(0)	🕒
$D \leq 66$	(3)	🕒	(0)	🕒	(0)	🕒	(0)	🕒	(0)	🕒
10	(**62**)	**1805s**	(**12**)	🕒	(**3**)	🕒	(0)	🕒	(0)	🕒
$D \leq 132$	(0)	🕒	(0)	🕒	(0)	🕒	(0)	🕒	(0)	🕒

Table 3: Results for infinite scheduling of DPAs. The first row in each age-instance is TAPAAL, the second line is UPPAAL TiGa. The format is (X) Ys where X the number of solved instances (within 3600 seconds) out of 100 and Y is the median time needed to solve the problem. The largest possible constant for each row is given as an upper bound of the deadline D.

	2 Yogurts/9 tokens		3 Yogurts/13 tokens	
Scale	UPPAAL	TAPAAL	UPPAAL	TAPAAL
1/6	1.10s	0.22s	🕒	95.44s
1/3	1.11s	5.64s	🕒	OOM
1/2	1.12s	42.68s	🕒	OOM
2/3	1.15s	231.29s	🕒	OOM
5/6	1.11s	656.10s	🕒	OOM
1/1	1.11s	OOM	🕒	OOM

Table 4: Results for the office fridge example from Figure 3 where constants are scaled by the given factor. We limit the number of tokens in the net to 9 or 13. Time is given in seconds, OOM signifies that the tool exceeded the memory-limitation (19 GB) and 🕒 indicates that more than one hour of computation time was used.

6.3 Office Fridge Example

As the last experiment, we return to our motivating example from Figure 3. In this experiment, we scale all constants in the model by the factors of $\frac{1}{6}$, $\frac{1}{3}$, $\frac{1}{2}$, $\frac{2}{3}$, $\frac{5}{6}$ and 1. We also scale the number of yogurts from 2 to 3—this also changes the weight on the transport-arc from *Fridge* to *Eat* to 3.

As illustrated in Table 4, our algorithm is sensitive to the size of the constants. This is expected as the algorithm uses an explicit exploration of the discrete state-space. We observe that eventually our algorithm runs out of memory—in particular with the exact values as provided in Figure 3. Compared to UPPAAL TiGa, it is apparent that the symbolic approach does not suffer from scaling the sizes of constants, however, it exceeds the one hour timeout for the case of 3 yogurts while we can still solve this problem for the scaling factor $\frac{1}{6}$.

7 Conclusion

We introduced timed-arc Petri net games and showed that for urgent controllers, the discrete and continuous-time semantics coincide. The presented discrete-time method for solving timed-arc Petri net games scales considerably better with the growing size of problems, compared to the existing symbolic methods. On the other hand, symbolic methods scale better with the size of the constants used in the model. In the future work, we may try to compensate for this drawback by using approximate techniques that "shrink" the constants to reasonable ranges while still providing conclusive answers in many cases, as demonstrated for pure reachability queries in [6]. Another future work includes the study of different synthesis objectives, as well as the generation

of continuous-time strategies from discrete-time analysis techniques on the subclass of urgent controllers.

Acknowledgments. The research leading to these results has received funding from the project DiCyPS funded by the Innovation Fund Denmark, the Sino Danish Research Center IDEA4CPS and the ERC Advanced Grant LASSO. The third author is partially affiliated with FI MU, Brno, Czech Republic.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, Apr. 1994.
2. M. Andersen, H. Larsen, J. Srba, M. Sørensen, and J. Taankvist. Verification of liveness properties on closed timed-arc Petri nets. In *Mathematical and Engineering Methods in Computer Science: 8th International Doctoral Workshop*, vol. 7721 of *LNCS*. Springer, 2013.
3. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In *Computer Aided Verification: 19th International Conference*, vol. 4590 of *LNCS*. Springer, 2007.
4. G. Behrmann, A. David, K. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks. Uppaal 4.0. In *Third International Conference on Quantitative Evaluation of Systems*, 2006.
5. B. Berthomieu and F. Vernadat. Time Petri nets analysis with TINA. In *Third International Conference on Quantitative Evaluation of Systems*. IEEE Computer Society, 2006.
6. S. Birch, T. Jacobsen, J. Jensen, C. Moesgaard, N. Samuelsen, and J. Srba. Interval abstraction refinement for model checking of timed-arc Petri nets. In *Formal Modeling and Analysis of Timed Systems: 12th International Conference*, vol. 8711 of *LNCS*. Springer, 2014.
7. T. Bolognesi, F. Lucidi, and S. Trigila. From Timed Petri Nets to Timed LOTOS. In *Protocol Specification, Testing and Verification X, Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol Specification*. North-Holland, 1990.
8. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Computer Aided Verification: 10th International Conference*, vol. 1427 of *LNCS*, 1998.
9. M. Bozga, O. Maler, and S. Tripakis. Efficient verification of timed automata using dense and discrete time semantics. In *Correct Hardware Design and Verification Methods: 10th IFIP WG10.5 Advanced Research Working Conference*, vol. 1703 of *LNCS*. Springer, 1999.
10. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *Concurrency Theory: 16th International Conference*, vol. 3653 of *LNCS*. Springer, 2005.
11. A. Church. Application of recursive arithmetic to the problem of circuit synthesis. *Journal of Symbolic Logic*, 1963.
12. A. Church. Logic, arithmetic, and automata. In *Proc. Internat. Congr. Mathematicians (Stockholm, 1962)*. Inst. Mittag-Leffler, 1963.
13. J. Cong, B. Liu, and Z. Zhang. Scheduling with soft constraints. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, Nov 2009.
14. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms. third edition., 2009.
15. A. David, L. Jacobsen, M. Jacobsen, K. Jørgensen, M. Møller, and J. Srba. TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In *Tools and Algorithms for the Construction and Analysis of Systems: 18th International Conference*, vol. 7214 of *LNCS*. Springer, 2012.
16. D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems: International Workshop*, vol. 407 of *LNCS*. Springer, 1990.
17. B. Finkbeiner. Bounded synthesis for Petri games. In *Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, vol. 9360 of *LNCS*. Springer, 2015.
18. B. Finkbeiner and E. Olderog. Petri games: Synthesis of distributed systems with causal memory. In *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification*, vol. 161 of *EPTCS*, 2014.
19. B. Finkbeiner and H. Peter. Template-based controller synthesis for timed systems. In *Tools and Algorithms for the Construction and Analysis of Systems: 18th International Conference*, vol. 7214 of *LNCS*. Springer, 2012.
20. G. Gardey, D. Lime, M. Magnin, and O. Roux. Romeo: A tool for analyzing time Petri nets. In *Computer Aided Verification: 17th International Conference*, vol. 3576 of *LNCS*. Springer, 2005.
21. H. Hanisch. Analysis of Place/Transition Nets with Timed Arcs and its Application to Batch Process Control. In *Application and Theory of Petri Nets 1993: 14th International Conference*, vol. 691 of *LNCS*. Springer, 1993.
22. A. Hoffman and J. Kruskal. Integral boundary points of convex polyhedra, in Linear Inequalities and Related Systems (H. Kuhn and A. Tucker, Eds.). *Annals of Maths. Study*, 1956.
23. S. Jacobs, R. Bloem, R. Brenguier, R. Könighofer, G. A. Pérez, J. Raskin, L. Ryzhyk, O. Sankur, M. Seidl, L. Tentrup, and A. Walker. The second reactive synthesis competition (SYNTCOMP 2015). In *Proceedings of the Fourth Workshop on Synthesis (SYNT'15)*, vol. 202 of *EPTCS*, 2016.
24. P. G. Jensen, K. G. Larsen, and J. Srba. Real-time strategy synthesis for timed-arc Petri net games via discretization. In *Proceedings of the 23rd International Symposium on Model Checking Software (SPIN'16)*, vol. 9641 of *LNCS*. Springer, 2016.
25. P. G. Jensen, K. G. Larsen, and J. Srba. PTrie: Data structure for compressing and storing sets via prefix sharing. In *Proceedings of the 14th International Colloquium on Theoretical Aspects of Computing (ICTAC'17)*, vol. 10580 of *LNCS*. Springer, 2017. To appear.
26. P. G. Jensen, K. G. Larsen, J. Srba, M. G. Sørensen, and J. H. Taankvist. Memory efficient data structures for explicit verification of timed systems. In *NASA Formal Methods: 6th International Symposium*, vol. 8430 of *LNCS*. Springer, 2014.
27. K. Jørgensen, K. G. Larsen, and J. Srba. Time-darts: A data structure for verification of closed timed automata.

In *Proceedings Seventh Conference on Systems Software Verification*, vol. 102 of *EPTCS*. Open Publishing Association, 2012.
28. J.-F. Kempf, M. Bozga, and O. Maler. As soon as probable: Optimal scheduling under stochastic uncertainty. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, vol. 7795 of *LNCS*. Springer, 2013.
29. K. G. Larsen and Y. Wang. Time-abstracted bisimulation: Implicit specifications and decidability. *Information and Computation*, 1997.
30. X. Liu and S. A. Smolka. Simple linear-time algorithms for minimal fixed points (extended abstract). In *Automata, Languages and Programming: 25th International Colloquium*, vol. 1443 of *LNCS*. Springer, 1998.
31. O. Maler, K. G. Larsen, and B. H. Krogh. On zone-based analysis of duration probabilistic automata. In *Proceedings of the 12th International Workshop on Verification of Infinite-State Systems (INFINITY'10)*, vol. 39 of *EPTCS*. Open Publishing Association, 2010.
32. J. Mateo, J. Srba, and M. Sørensen. Soundness of timed-arc workflow nets in discrete and continuous-time semantics. *Fundamenta Informaticae*, 2015.
33. H. Peter. Component-based abstraction refinement for timed controller synthesis. In *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, IEEE Computer Society, 2009.
34. H. Peter, R. Ehlers, and R. Mattmüller. Synthia: Verification and synthesis for timed automata. In *Computer Aided Verification: 23rd International Conference*, vol. 7214 of *LNCS*. Springer, 2011.
35. A. Pnueli, E. Asarin, O. Maler, and J. Sifakis. Controller synthesis for timed automata. In *System Structure and Control*. Citeseer, Elsevier, 1998.
36. J. Raskin, M. Samuelides, and L. Begin. Petri games are monotone but difficult to decide. Technical report, Université Libre De Bruxelles, 2003.
37. V. Ruiz, F. C. Gomez, and D. de Frutos Escrig. On non-decidability of reachability for timed-arc Petri nets. In *The 8th International Workshop on Petri Nets and Performance Models. Proceedings.*, 1999.
38. Q. Zhou, M. Wang, and S. Dutta. Generation of optimal control policy for flexible manufacturing cells: A Petri net approach. *The International Journal of Advanced Manufacturing Technology*, 1995.