Real-Time Strategy Synthesis for Timed-Arc Petri Net Games via Discretization

Peter Gjøl Jensen(✉), Kim Guldstrand Larsen, and Jiří Srba

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark
{pgj,kgl,srba}@cs.aau.dk

Abstract. Automatic strategy synthesis for a given control objective can be used to generate correct-by-construction controllers of 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 been traditionally 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 controller correct-by-construction, as already proposed by Church [12,13], is a more difficult problem as illustrated by the SYNTCOMP competition and SYNT workshop [1]. This area has recently seen renewed interest, partly given the rise in computational power that makes the synthesis feasible. We focus on the family of timed systems, where for the model of timed automata [2] synthesis has already been proposed [33] and implemented [4,11].

In the area of model checking, symbolic continuous-time on-the-fly methods were ensuring the success of tools such as Kronos [9], UPPAAL [5], Tina [6]

D. Bošnački and A. Wijs (Eds.): SPIN 2016, LNCS 9641, pp. 129–146, 2016.
DOI: 10.1007/978-3-319-32582-8_9

and Romeo [21], utilizing the zone abstraction approach [2] via the data structure DBM [16]. These symbolic techniques were recently employed in on-the-fly algorithms [28] for synthesis of controllers for timed games [4,11,33]. 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 [3], in particular in combination with additional techniques as time-darts [25], constant-reducing approximation techniques [7] and memory-preserving data structures as PTrie [24].

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 [8,23] and we show that in the general setting the existence of a controller for a 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 timed models [10], in particular also for timed-arc Petri nets [30]. 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 the tool TAPAAL [15], including the memory optimization technique via PTrie [24], and the experimental data show a promising performance on a large data-set of infinite job scheduling problems as well as on other examples.

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. [11] and later implemented in the tool UPPAAL TiGa [4]. This work is based on the symbolic verification techniques invented by Alur and Dill [2] in combination with ideas on synthesis by Pnueli et al. [33] and on-the-fly dependency graph algorithms suggested by Liu and Smolka [28]. For timed games, abstraction refinement approaches have been proposed and implemented by Peter et al. [31,32] and Finkbeiner and Peter [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. [3]. As an additional contribution, we implement our solution in the tool TAPAAL, utilizing memory reduction techniques by Jensen et al. [24], and compare the performance of both discrete-time and continuous-time techniques.

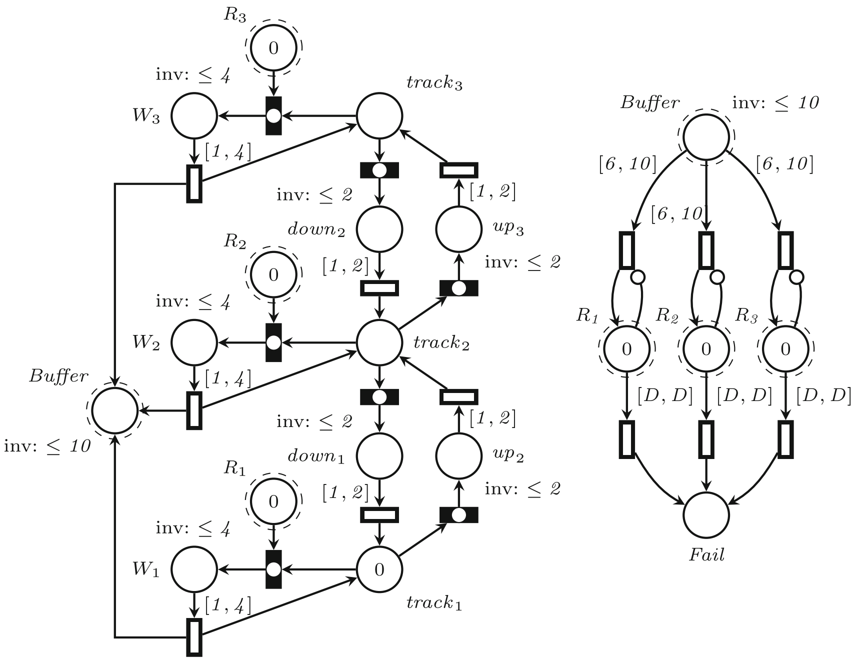


Fig. 1. A timed-arc Petri net game model of a harddisk

Control synthesis and supervisory control was also studied for the family of Petri net models [17,18,34,36] but these works do not consider the timing aspects.

2 Motivating Example of Disk Operation Scheduling

We shall now provide an intuitive description of the timed-arc Petri net game of *disk operation scheduling* in Fig. 1, modelling the scheduler of a mechanical harddisk 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 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 an *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 1 ms to 4 ms (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 1 ms to 2 ms). 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 6 ms to 10 ms. 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. 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.

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 $(S, Act, \rightarrow)$ where S is the set of states, Act is the set of actions and $\rightarrow \subseteq S \times (Act \cup \mathbb{R}^{\geq 0}) \times S$ is the transition relation written as $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$. 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 $\mathcal{I} \overset{\text{def}}{=} \{[a, b] \mid a \in \mathbb{N}_0, b \in \mathbb{N}_0^\infty, a \leq b\}$ and its subset $\mathcal{I}^{\text{inv}} \overset{\text{def}}{=} \{[0, b] \mid b \in \mathbb{N}_0^\infty\}$ 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*,*
- $g : IA \rightarrow \mathcal{I}$ *is a* time constraint function *assigning guards to input arcs such that*
  - *if* $(p, t) \in IA$ *and* $t \in T_{urg}$ *then* $g((p, t)) = [0, \infty]$*,*
- $w : IA \cup OA \rightarrow \mathbb{N}$ *is a function assigning* weights *to input and output arcs,*
- $Type : IA \cup OA \rightarrow \mathbf{Types}$ *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* $w((p, t)) = w((t, p'))$*,*
- $I : P \rightarrow \mathcal{I}^{inv}$ *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].

Before we give the formal semantics of the model, let us fix some notation. Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ be a TAPN. We denote by ${}^\bullet x \overset{\text{def}}{=} \{y \in P \cup T \mid (y, x) \in IA \cup OA,\ Type((y, x)) \neq Inhib\}$ the preset of a transition or a place x. Similarly, the postset is defined as $x^\bullet \overset{\text{def}}{=} \{y \in P \cup T \mid (x, y) \in (IA \cup OA)\}$. 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). *Let $N = (P, T, T_{urg}, IA, OA, g, w, Type, I)$ 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.*

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

– *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.*

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

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

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

A given TAPN N defines a TTS $T(N) \stackrel{\text{def}}{=} (\mathcal{M}(N), T, \rightarrow)$ 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 *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 $\rightarrow \stackrel{\text{def}}{=} \bigcup_{t \in T} \xrightarrow{t} \cup \bigcup_{d \in \mathbb{R}^{\geq 0}} \xrightarrow{d}$. By $M \xrightarrow{d,t} M'$ we denote that there is a marking M'' such that $M \xrightarrow{d} M'' \xrightarrow{t} M'$.

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*.

3.1 Timed-Arc Petri Net Game

We shall now extend the TAPN model into the game setting by partitioning the set of 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 controller T_{ctrl} and environment T_{env} sets.*

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

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

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 $d'' \in \mathbb{R}^{\geq 0}$ for all $t \in T_{ctrl}$ we have that if $M' \xrightarrow{d''} M''$ then $M'' \overset{t}{\not\rightarrow}$, and
- if $\sigma(M) = M'$ then there is $d \in \mathbb{R}^{\geq 0}$ and there is $t \in T_{ctrl}$ where $M \xrightarrow{d,t} M'$.

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 $\pi = M_1 M_2 \ldots M_n \ldots \in \mathcal{M}(G)^\omega$ 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 $M \circ \pi = M M_1 \ldots M_n \ldots$ 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 $\mathbb{P}_\sigma : \mathcal{M}(G) \rightarrow 2^{\mathcal{M}(G)^\omega}$ returning for a given marking M the set of all possible plays starting from M under the strategy σ.*

- *If $\sigma(M) = wait$ then $\mathbb{P}_\sigma(M) = \{M \circ \mathbb{P}_\sigma(M') \mid d \in \mathbb{R}^{\geq 0},\ t \in T_{env},\ M \xrightarrow{d,t} M'\} \cup X$ 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 $M' \overset{d''}{\not\rightarrow}$ for any $d'' > 0$ and $M' \overset{t}{\not\rightarrow}$ 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 $\mathbb{P}_\sigma(M) = \{M \circ \mathbb{P}_\sigma(\sigma(M))\} \cup \{M \circ \mathbb{P}_\sigma(M') \mid d' \leq d, t' \in T_{env}, M \xrightarrow{d',t'} M'\}$.*

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 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 that are boolean expressions 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*

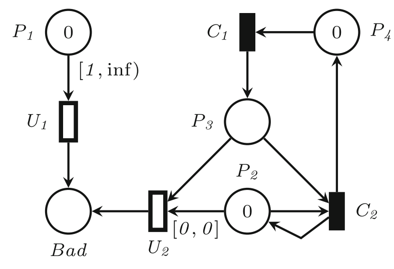
$$\forall \pi \in \mathbb{P}_\sigma(M_0). \forall M \in \pi. M \models \varphi. \tag{1}$$

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

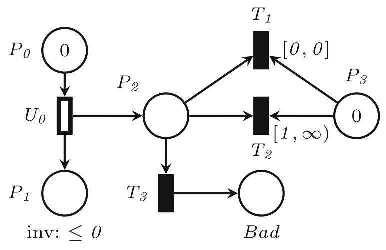
4 Controller Synthesis in Continuous vs. Discrete Time

It is known that for classical TAPNs the continuous and discrete-time semantics coincide up to reachability [30], which is what safety synthesis reduces to if the set of controllable transitions is empty. Contrary to this, Fig. 2a and b show that this does not hold in general for safety strategies.

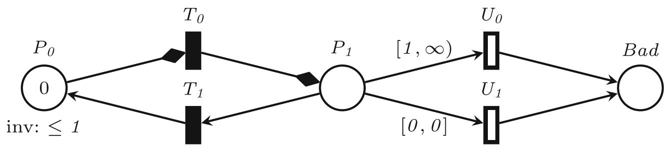
For the game in Fig. 2a, there exists a strategy for the controller and the safety objective $Bad \leq 0$ but this is the case only in the continuous-time semantics as the controller has to keep the age of the token in place P_1 strictly below 1, otherwise the environment can mark the place Bad by firing U_1. However, if the controller fires transition C_1 without waiting, U_2 becomes enabled and the environment can again break the safety. Hence it is impossible to find a discrete-time strategy as even the smallest possible delay of 1 time unit will enable U_1. However, if the controller waits an infinitesimal amount (in the continuous



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



(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 but not under the discrete-time semantics (without exploiting Zeno behavior).

Fig. 2. Difference between continuous and discrete-time semantics

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 that do not reset the age of tokens (depicted by arrows with diamond-headed tips), as demonstrated in Fig. 2c. Here the only winning 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 lemma.

Lemma 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 2b 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 one where 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), however then T_2 will be enabled. Hence the controller can in both cases avoid the firing of T_3 in the discrete-time semantics. This implies the following lemma.

Lemma 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 the fact 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. We say that a given TAPG is with an *urgent controller* if all controllable transitions are urgent, formally $T_{ctrl} \subseteq T_{urg}$.

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.*

Proof (Sketch). The existence of a winning controller strategy in the continuous-time semantics clearly implies the existence of such a strategy also in the discrete-time because here the environment is restricted to playing only integer delays and the controller can always react to these according to the continuous-time strategy that exists by our assumption. Because the controller is making only urgent choices or waits for the next environmental move, all transitions happen in the discrete-time points.

For the other direction, we prove the converse via the use of linear programming as used e.g. in [30]. Assuming that the urgent controller does not have a winning strategy in the continuous-time semantics, we will argue that the controller does not have a winning strategy in the discrete-time semantics either. Due to the assumption, we know that the environment can in any current marking choose a real-time delay and an uncontrollable transition in such a way that irrelevant of what the controller chooses, it eventually reaches a marking violating the safety condition φ. Such an environmental strategy can be described as a finite tree where nodes are markings, edges contain the information about the delay and transition firing, the branching describes all controller choices and each leaf of the tree is a marking that satisfies $\neg\varphi$. The existence of this environmental strategy follows from the determinacy of the game that guarantees that one of the players must have a winning strategy (to see this, we realize that the environmental strategy contains only finite branches, all of them ending in a marking satisfying $\neg\varphi$, and hence we have an instance of an open game that is determined by the result of Gale and Stewart [20]—see also [22]).

As we assume that the environment can win in the continuous-time semantics, the delays in the tree may be nonnegative real numbers (controller's moves in the tree are always with delay 0). Our aim is to show that there is another winning tree for the environment, however, with integer delays only. This can be done by replacing the delays in the tree by variables and reformulating the firing conditions of the transitions in the tree as a linear program. Surely, the constraints in the linear program have, by our assumption, a nonnegative real solution. Moreover, the constraint system uses only closed difference constraints (nonstrictly bounding the difference of two variables from below or above) and we can therefore reduce the linear program to a shortest-path problem with integer weights only and this implies that an integer solution exists too [14]. This means that there is a tree describing an environmental winning strategy using only integer delays and hence the controller does not have a winning strategy in the discrete-time setting. The technical details of the proof are provided in the full version of the paper. □

5 Discrete-Time Algorithm for Controller Synthesis

We shall now define the discrete-time algorithm for synthesizing 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 [35] for TAPNs).

We address the undecidability issue by enforcing a given 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.

5.1 Extrapolation of TAPGs

We shall now recall a few results from [3] 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 hold also for continuous-time semantics, however, the finiteness of the extrapolated state space is not guaranteed in this case.

Let $G = (P, T, T_{env}, T_{ctrl}, T_{urg}, \mathit{IA}, \mathit{OA}, g, w, \mathit{Type}, I)$ be a TAPG. In [3] 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 [3]). 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 $|M_>(p)| = |M'_>(p)|$. In other words M and M' agree on the tokens with ages below the maximum constants and have the same number of tokens above the maximum constant.

The relation $\equiv$ is an equivalence relation and it is also a timed bisimulation (see e.g. [27]) where 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.

Theorem 2 ([3]). *The relation $\equiv$ is a timed bisimulation.*

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

Definition 6 (Cut). *Let M be a marking. We define its canonical marking $cut(M)$ by* $cut(M)(p) = M_\leq(p) \uplus \{\underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_>(p)| \text{ times}}\}$.

Lemma 3 ([3]). *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 $cut(M_1) = cut(M_2)$.*

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* for storing markings to be explored, *Losing* that contains marking where such a strategy does not exist, *Depend* for maintaining the set of dependencies to be reinserted to the waiting list whenever a marking is declared as losing, and *Processed* 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.

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

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 $\varphi$ and a bound $k$.
Output: $tt$ if there exists a controller strategy ensuring $\varphi$ 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         $Succs_{delay} := \begin{cases} \emptyset & \text{if } M \overset{1}{\not\to} \\ \{cut(M')\} & \text{if } M \xrightarrow{1} M' \end{cases}$
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 $ff$
```

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 [4] version 0.18, the state-of-the-art continuous-time model checker for timed games. The experiments were run on AMD Opteron

Table 1. Time in seconds to find a controller strategy for the disk operation scheduling for the smallest D where such a strategy exists.

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	30.14s	69.78s	128.58s	216.44s	316.71s	491.65s	665.34s
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	1.98s	7.34s	30.73s	101.92s	210.25s	398.00s	768.11s
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	2.20s	16.52s	72.41s	244.28s	885.60s	(2132.71s)	🕑
UPPAAL	885.56s	🕑	🕑	🕑	🕑	🕑	🕑

6376 processor limited to using 16 GB of RAM[1] and with one hour timeout (denoted by 🕑).

6.1 Disk Operation Scheduling

In the disk operation scheduling model presented in Sect. 2 we scale the problem by changing the number of tracks and the number of simultaneous read streams. A similar 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 1 show that our algorithm scales considerably better than TiGa (that suffers from the large fragmentation of zone federations) as the number of tracks increases and it is significantly better when we add more read streams (and hence increase the concurrency and consequently the number of timed tokens/clocks).

6.2 Infinite Job Shop Scheduling

In our second experiment, infinite job shop scheduling, we consider the duration probabilistic automata [29]. Kempf et al. [26] 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

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

Table 2. 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 s) 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 Processes/7-13 tokens

Max Age	10 Tasks	12 Tasks	14 Tasks	16 Tasks	18 Tasks
5	(100) **63s**	(**100**) **141s**	(**100**) **283s**	(**100**) **570s**	(**100**) **829s**
$D \leq 144$	(100) 100s	(98) 413s	(85) 1201s	(35) 🕒	(18) 🕒
10	(**100**) 318s	(**100**) **882s**	(**96**) **1555s**	(**65**) **2911s**	(**14**) 🕒
$D \leq 288$	(96) **221s**	(69) 1443s	(43) 🕒	(16) 🕒	(1) 🕒
15	(**99**) 1054s	(**78**) 2521s	(19) 🕒	(**14**) 🕒	(**2**) 🕒
$D \leq 432$	(87) **315s**	(60) **1960s**	(19) 🕒	(8) 🕒	(0) 🕒
20	(80) 2479s	(22) 🕒	(14) 🕒	(3) 🕒	(**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**) **39s**	(**99**) **402s**	(**66**) **1884s**	(**38**) 🕒
$D \leq 57$	(99) 16s	(69) 1827s	(4) 🕒	(0) 🕒	(0) 🕒
10	(**100**) **15s**	(**97**) **484s**	(**47**) 🕒	(**20**) 🕒	(**6**) 🕒
$D \leq 114$	(98) 32s	(52) 3338s	(6) 🕒	(0) 🕒	(0) 🕒
15	(**100**) 51s	(**69**) **1373s**	(**28**) 🕒	(**4**) 🕒	(0) 🕒
$D \leq 171$	(98) **27s**	(50) 🕒	(1) 🕒	(0) 🕒	(0) 🕒

4 Processes/13-25 tokens

Max Age	2 Tasks	3 Tasks	4 Tasks	5 Tasks	6 Tasks
5	(**92**) **215s**	(**30**) 🕒	(**7**) 🕒	(**1**) 🕒	(0) 🕒
$D \leq 66$	(3) 🕒	(0) 🕒	(0) 🕒	(0) 🕒	(0) 🕒
10	(**60**) **2286s**	(**11**) 🕒	(**2**) 🕒	(0) 🕒	(0) 🕒
$D \leq 132$	(0) 🕒	(0) 🕒	(0) 🕒	(0) 🕒	(0) 🕒

environment). We consider here a variant of this problem that should guarantee an infinite (cyclic) scheduling where all processes that share various resources and 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 now to find a schedule such that all processes always meet their deadline. The problem can be modelled using urgent controller, so the discrete and continuous-time semantics again 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 (excepted 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 2 shows a similar trend as in 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.

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 [7]. 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 EU FP7 FET projects CASSTING and SENSATION, 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. SYNT 2015. Electronic Proceedings in Theoretical Computer Science (2015). http://formal.epfl.ch/synt/2015/
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoret. Comput. Sci. **126**(2), 183–235 (1994)
3. Andersen, M., Larsen, H.G., Srba, J., Sørensen, M.G., Haahr Taankvist, J.: Verification of liveness properties on closed timed-arc Petri nets. In: Kučera, A., Henzinger, T.A., Nešetřil, J., Vojnar, T., Antoš, D. (eds.) MEMICS 2012. LNCS, vol. 7721, pp. 69–81. Springer, Heidelberg (2013)
4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-TiGa: time for playing games!. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
5. Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: Third International Conference on Quantitative Evaluation of Systems, pp. 125–126 (2006)
6. Berthomieu, B., Vernadat, F.: Time Petri nets analysis with TINA. In: Third International Conference on Quantitative Evaluation of Systems, pp. 123–124. IEEE Computer Society (2006)

7. Birch, S.V., Jacobsen, T.S., Jensen, J.J., Moesgaard, C., Nørgaard Samuelsen, N., Srba, J.: Interval abstraction refinement for model checking of timed-arc Petri nets. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 237–251. Springer, Heidelberg (2014)
8. Bolognesi, T., Lucidi, F., Trigila, S.: From timed Petri nets to timed LOTOS. In: Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol Specification, Testing and Verification X, North-Holland, pp. 395–408 (1990)
9. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: a model-checking tool for real-time systems. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 546–550. Springer, Heidelberg (1998)
10. Bozga, M., Maler, O., Tripakis, S.: Efficient verification of timed automata using dense and discrete time semantics. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 125–141. Springer, Heidelberg (1999)
11. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005)
12. Church, A.: Application of recursive arithmetic to the problem of circuit synthesis. J. Symbolic Logic **28**(4), 289–290 (1963)
13. Church, A.: Logic, arithmetic, and automata. In: Proceedings of the International Congress of Mathematicians (Stockholm, 1962), pp. 23–35. Institute Mittag-Leffler (1963)
14. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition, 3rd edn. The MIT Press (2009). ISBN: 0262033844 9780262033848
15. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAL 2.0: integrated development environment for timed-arc Petri nets. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 492–497. Springer, Heidelberg (2012)
16. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
17. Finkbeiner, B.: Bounded synthesis for Petri games. In: Meyer, R., et al. (eds.) Olderog-Festschrift. LNCS, vol. 9360, pp. 223–237. Springer, Heidelberg (2015)
18. Finkbeiner, B., Olderog, E.: 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, pp. 217–230 (2014)
19. Finkbeiner, B., Peter, H.-J.: Template-based controller synthesis for timed systems. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 392–406. Springer, Heidelberg (2012)
20. Gale, D., Stewart, F.M.: Infinite games with perfect information. In: Contributions to the Theory of Games. Annals of Mathematics Studies, no. 28, vol. 2, pp. 245–266. Princeton University Press, Princeton, N.J (1953)
21. Gardey, G., Lime, D., Magnin, M., Roux, O.H.: Romeo: a tool for analyzing time Petri nets. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 418–423. Springer, Heidelberg (2005)
22. Gurevich, Y.: Games people play. In: Lane, S.M., Siefkes, D. (eds.) The Collected Works of J. Richard Büchi, pp. 517–524. Springer, Berlin (1990)
23. Hanisch, H.-M.: Analysis of place/transition nets with timed arcs and its application to batch process control. In: Marsan, M.A. (ed.) ICATPN 1993. LNCS, vol. 691, pp. 282–299. Springer, Heidelberg (1993)

24. Jensen, P.G., Larsen, K.G., Srba, J., Sørensen, M.G., Taankvist, J.H.: Memory efficient data structures for explicit verification of timed systems. In: Badger, J.M., Rozier, K.Y. (eds.) NFM 2014. LNCS, vol. 8430, pp. 307–312. Springer, Heidelberg (2014)
25. Jørgensen, K.Y., Larsen, K.G., Srba, J.: Time-darts: a data structure for verification of closed timed automata. In: Proceedings Seventh Conference on Systems Software Verification, vol. 102 of EPTCS, pp. 141–155. Open Publishing Association (2012)
26. Kempf, J.-F., Bozga, M., Maler, O.: As soon as probable: optimal scheduling under stochastic uncertainty. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 385–400. Springer, Heidelberg (2013)
27. Larsen, K.G., Wang, Y.: Time-abstracted bisimulation: implicit specifications and decidability. Inf. Comput. **134**(2), 75–101 (1997)
28. Liu, X., Smolka, S.A.: Simple linear-time algorithms for minimal fixed points (extended abstract). In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 53–66. Springer, Heidelberg (1998)
29. Maler, O., Larsen, K.G., Krogh, B.H.: On zone-based analysis of duration probabilistic automata. In: Proceedings 12th International Workshop on Verification of Infinite-State Systems, vol.39 of EPTCS, pp. 33–46 (2010)
30. Mateo, J.A., Srba, J., Sørensen, M.G.: Soundness of timed-arc workflow nets in discrete and continuous-time semantics. Fundamenta Informaticase **140**(1), 89–121 (2015)
31. Peter, H.: Component-based abstraction refinement for timed controller synthesis. In: IEEE, pp. 364–374. IEEE Computer Society (2009)
32. Peter, H.-J., Ehlers, R., Mattmüller, R.: Synthia: verification and synthesis for timed automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 649–655. Springer, Heidelberg (2011)
33. Pnueli, A., Asarin, E., Maler, O., Sifakis, J.: Controller synthesis for timed automata. System Structure and Control. Citeseer, Elsevier (1998)
34. Raskin, J.F., Samuelides, M., Begin, L.V.: Petri games are monotone but difficult to decide. Technical report, Université Libre De Bruxelles (2003)
35. Ruiz, V.V., Cuartero Gomez, F., de Frutos Escrig, D.: On non-decidability of reachability for timed-arc Petri nets. In: Proceedings of the 8th International Workshop on Petri Nets and Performance Models, pp. 188–196 (1999)
36. Zhou, Q., Wang, M., Dutta, S.P.: Generation of optimal control policy for flexible manufacturing cells: a Petri net approach. Int. J. Adv. Manuf. Technol. **10**(1), 59–65 (1995)