Verification of Timed-Arc Petri Nets

Lasse Jacobsen, Morten Jacobsen, Mikael H. Møller, and Jiří Srba*

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

Abstract. Timed-Arc Petri Nets (TAPN) are an extension of the classical P/T nets with continuous time. Tokens in TAPN carry an age and arcs between places and transitions are labelled with time intervals restricting the age of tokens available for transition firing. The TAPN model posses a number of interesting theoretical properties distinguishing them from other time extensions of Petri nets. We shall give an overview of the recent theory developed in the verification of TAPN extended with features like read/transport arcs, timed inhibitor arcs and age invariants. We will examine in detail the boundaries of automatic verification and the connections between TAPN and the model of timed automata. Finally, we will mention the tool TAPAAL that supports modelling, simulation and verification of TAPN and discuss a small case study of alternating bit protocol.

1 Introduction

Formal verification of embedded and hybrid systems is an active research area. Recently, a lot of attention has been devoted to the analysis of systems with quantitative attributes like timing, cost and probability. In particular, several different time-dependent models were developed over the two last decades or so. These models are often introduced as a time extension of some well-studied untimed formalism and include, among others, (networks of) timed automata [7, 8] and different time extensions of the Petri net model [45]. These formalisms are nowadays supported by a number of tools [1, 2, 12, 19, 20, 26, 29, 36] and exploited in model-driven system design methodologies.

We shall focus on the Petri net model extended with continuous time. The timing aspects are associated with different parts of the model in the various time-extended Petri net formalisms. For example, *timed transitions Petri nets* where transitions are annotated with their durations were proposed in [46]. A model in which time parameters are associated with places is called *timed places Petri nets* and it was introduced in [51]. *Time Petri nets* of Merlin and Faber [38, 39] were introduced in 1976 and associate time intervals to each transition. The intervals define the earliest and latest firing time of the transition since it became enabled. Yet another model of *timed-arc Petri nets* was first studied around 1990 by Bolognesi, Lucidi, Trigila and Hanisch [15, 28]. Here time information is

* Partially supported by Ministry of Education of the Czech Republic, project No. MSM 0021622419.

attached to the tokens in the net representing their relative age while arcs from places to transition contain time intervals that restrict the enableness of the transitions. For an overview of the different extensions see e.g. [18, 44, 54, 55].

In this paper we will survey the results and techniques connected with timed-arc Petri nets (TAPN). This model is particularly suitable for modelling of manufacturing systems, work-flow management and similar applications [4, 5, 42, 43, 49, 50] and a recently developed tool TAPAAL [22] enables automatic verification of bounded TAPNs extended with transport/inhibitor arcs and age invariants.

The outline of the paper is as follows. In Section 2 we give an informal introduction to the TAPN model and in Section 3 we describe its formal syntax and semantics. Section 4 illustrates the modeling of alternating bit protocol as a TAPN and Section 5 explains the main decidability and complexity results. Sections 6 and 7 define the TCTL logic and explain a translation from TAPN with transport arcs, inhibitor arcs and age invariants to UPPAAL timed automata. The translation preserves TCTL model checking including liveness properties. Finally, Section 8 gives a short conclusion.

2 Informal Introduction to Timed-Arc Petri Nets

We shall first informally introduce the TAPN model extended with transport arcs, age invariants and inhibitor arcs. A basic timed-arc Petri net is presented in Figure 1a. It consists of two transitions t_1 and t_2 drawn as rectangels and five places $p_1, \ldots, p_5$ drawn as circles. There is one token of age 0.0 in each of the places p_1, p_2 and p_3. Initially, only the transition t_1 is enabled because its input place p_1 contains a token of an age that fits into the interval $[0, \infty]$ present on the arc from p_1 to t_1. The transition t_2 requires a token of any age in p_2 but also a token of an age in the interval $[4, 5]$ in p_3. This is why t_2 is not enabled at the moment. Because t_1 is enabled, it can fire, whereby it removes the token from p_1 and produces a new fresh token of age 0.0 in each of the output places p_4 and p_5. Instead, it is also possible that the net performs a time delay of, say, 4.5 time units. By this step all tokens in the net grow 4.5 time units older. As all tokens are now of age 4.5, both t_1 and t_2 are enabled and can fire. Notice that the tokens that are produced even after the time delay are of age 0.0.

Let us now introduce *transport arcs* into our example net. Specifically, we will replace the normal arcs from p_1 to t_1 and from t_1 to p_5 with a pair of transport arcs with solid arrow tips as illustrated in Figure 1b. Transport arcs come always in pairs like this. Note that the symbol ':1' on the transport arcs is there to denote the pairing of the arcs (as it is in general possible to have more than one pair of transport arcs connected to a transition).

For the sake of illustration, assume that we have initially made a time delay of 2.5 time units such that all tokens are now of age 2.5. Transition t_1 is still the only enabled one in the net at this point, however, there is a difference when we fire t_1. Firing t_1 will remove a token of age 2.5 from p_1 and produce a token of age 0.0 in the place p_4 as before. However, due to the transport arcs, it will

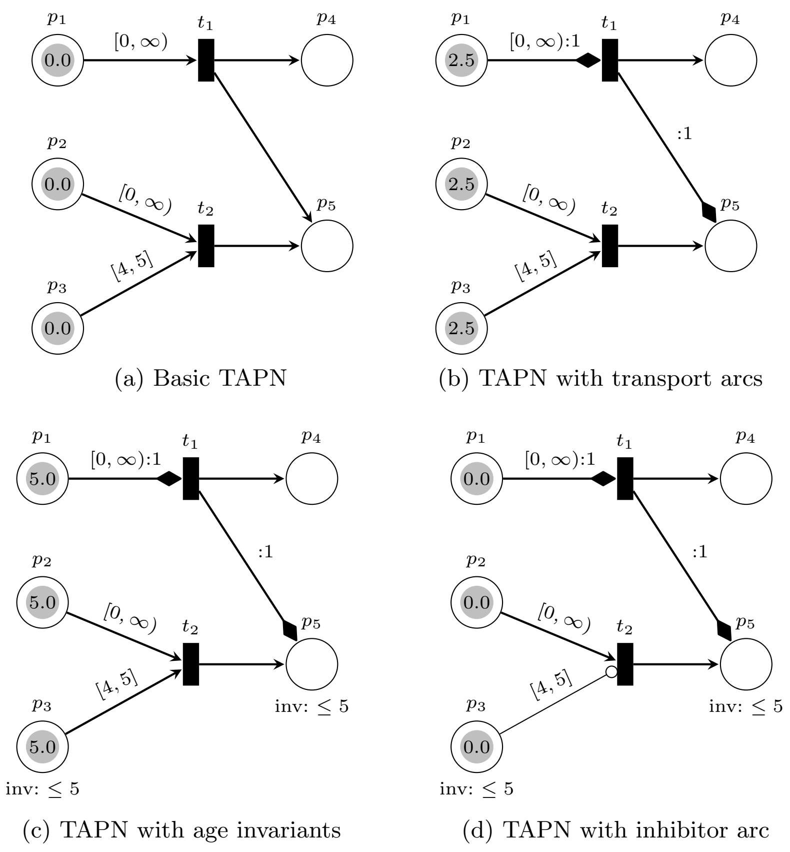


Fig. 1: Examples of timed-arc Petri nets

produce a token at p_5 of the same age as the token it removed from p_1, i.e. of the age 2.5 in this case. Hence transport arcs allow us to preserve the ages of tokens as they are transported through the net. This is a feature particularly suitable for modelling of e.g. product lines where we need to track the age of a product from its start until its final production stage.

The next modelling feature is called *age invariant*. It simply restricts the maximum age of tokens that can appear in certain places. In our running example, we can add age invariants to the places p_3 and p_5 as illustrated in Figure 1c. These invariants will disallow tokens older than 5 time units in these two places.

Assume a situation after a time delay such that all tokens are of age 5.0 as in the figure. At this point no further time delays are possible as they would violate the invariant at p_3. We are thus forced to fire either t_1 or t_2, both of which are enabled. Invariants hence facilitate the modelling of urgency. One of the particularities of the combination of invariants and transport arcs is that transitions are disabled should their transport arcs move a token to a place where the age of the token violates the age invariant. In our example t_1 is enabled but should the invariant at p_5 allow tokens only of age at most 4, then it would be disabled.

Finally, we introduce *inhibitor arcs* that disable the firing of a transition based on the presence of tokens in certain places. In our example we will replace the arc from p_3 to t_2 with an inhibitor arc with circle arrow tip as illustrated in Figure 1d. Transition t_2 is then blocked whenever there is a token in place p_3 with age in the interval $[4, 5]$. As this is not the case in the depicted situation, both t_1 and t_2 are enabled and can be fired. Firing of t_2 has no effect on the tokens in the place p_3. If instead a time delay of 4 time units was performed, t_2 would be blocked and only t_1 could fire. This concludes the informal introduction to timed-arc Petri nets.

3 Formal Definition of Timed-Arc Petri Nets

We start with the preliminaries and the definition of timed transition system.

We let $\mathbb{N}_0$ and $\mathbb{R}_{\geq 0}$ denote the sets of nonnegative integers and nonnegative real numbers, respectively.

A *timed transition system* (TTS) is a pair $T = (S, \longrightarrow)$ where S is a set of states (or processes) and $\longrightarrow \subseteq S \times S \cup S \times \mathbb{R}_{\geq 0} \times S$ is a transition relation.

We write $s \longrightarrow s'$ whenever $(s, s') \in \longrightarrow$ and call them *discrete transitions*, and $s \xrightarrow{d} s'$ whenever $(s, d, s') \in \longrightarrow$ and call them *delay transitions*. We require that all TTS we consider satisfy the following standard axioms for delay transitions (see e.g. [13]). For all $d, d' \in \mathbb{R}_{\geq 0}$ and $s, s', s'' \in S$:

1. **Time Additivity:** if $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$ then $s \xrightarrow{d+d'} s''$,
2. **Time Continuity:** if $s \xrightarrow{d+d'} s''$ then $s \xrightarrow{d} s' \xrightarrow{d'} s''$ for some s',
3. **Zero Delay:** $s \xrightarrow{0} s$ for each state s, and
4. **Time Determinism:** if $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$ then $s' = s''$.

By $s[d]$ we denote the state s' (if it exists) such that $s \xrightarrow{d} s'$. Time determinism ensures the uniqueness of $s[d]$. We write $s \Longrightarrow s'$ if $s \longrightarrow s'$ or $\xrightarrow{d} s'$ for some d. The notation $\Longrightarrow^*$ denotes the reflexive and transitive closure of $\Longrightarrow$.

3.1 Syntax

We shall now define the timed-arc Petri net model. First, we define the set of well-formed time intervals by the abstract syntax where $a, b \in \mathbb{N}_0$ and $a < b$:

$$I ::= [a, a] \mid [a, b] \mid [a, b) \mid (a, b] \mid (a, b) \mid [a, \infty) \mid (a, \infty) \quad .$$

We denote the set of all well-formed time intervals by $\mathcal{I}$. Further, the set of all well-formed time intervals for invariants is denoted by $\mathcal{I}^{inv}$ and defined according to the following abstract syntax:

$$I_{\text{Inv}} ::= [0, 0] \mid [0, b] \mid [0, b) \mid [0, \infty) \quad .$$

The predicate $r \in I$ is defined for $r \in \mathbb{R}_{\geq 0}$ in the expected way.

Definition 1. *A* TAPN *is a 7-tuple* $(P, T, \mathit{IA}, \mathit{OA}, \mathit{Transport}, \mathit{Inhib}, \mathit{Inv})$*, where*

– P *is a finite set of* places*,*
– T *is a finite set of* transitions *s.t.* $P \cap T = \emptyset$*,*
– $\mathit{IA} \subseteq P \times \mathcal{I} \times T$ *is a finite set of* input arcs *s.t.*

$$((p, I, t) \in \mathit{IA} \wedge (p, I', t) \in \mathit{IA}) \Rightarrow I = I'$$

– $\mathit{OA} \subseteq T \times P$ *is a finite set of* output arcs*,*
– $\mathit{Transport} : \mathit{IA} \times \mathit{OA} \rightarrow \{\mathit{true}, \mathit{false}\}$ *is a function defining* transport arcs *which are pairs of input and output arcs connected to some transition, formally we require that for all* $(p, I, t) \in \mathit{IA}$ *and* $(t', p') \in \mathit{OA}$ *whenever* $\mathit{Transport}((p, I, t),(t', p'))$ *then* $t = t'$ *and moreover for all* $\alpha \in \mathit{IA}$ *and all* $\beta \in \mathit{OA}$

$$(\mathit{Transport}(\alpha, (t', p')) \Rightarrow\ \alpha = (p, I, t)) \wedge$$
$$(\mathit{Transport}((p, I, t), \beta) \Rightarrow\ \beta = (t', p'))$$

– $\mathit{Inhib} : \mathit{IA} \longrightarrow \{\mathit{true}, \mathit{false}\}$ *is a function defining* inhibitor arcs *which do not collide with transport arcs, i.e. whenever* $\mathit{Transport}(\alpha, \beta)$ *for some* $\alpha \in \mathit{IA}$ *and* $\beta \in \mathit{OA}$ *then* $\neg \mathit{Inhib}(\alpha)$*, and*
– $\mathit{Inv} : P \rightarrow \mathcal{I}^{inv}$ *is a function assigning* age invariants *to places.*

A TAPN is called basic *if the functions Transport and Inhib return false for all arcs.*

The preset of a transition $t \in T$ is defined as ${}^\bullet t = \{p \in P \mid (p, I, t) \in \mathit{IA}\}$. Similarly, the postset of t is defined as $t^\bullet = \{p \in P \mid (t, p) \in \mathit{OA}\}$. For technical convenience we do not allow multiple arcs.

3.2 Semantics

We will now define the semantics of the TAPN. First we define a marking, which is a function assigning to each place a finite multiset of nonnegative real numbers (all such finite multisets are denoted by $\mathcal{B}(\mathbb{R}_{\geq 0})$). The real numbers represent the age of tokens that are currently at a given place; the age of every token must moreover respect the age invariant of the place where the token is located.

Definition 2 (Marking). *Let* $N = (P, T, \mathit{IA}, \mathit{OA}, \mathit{Transport}, \mathit{Inhib}, \mathit{Inv})$ *be a TAPN. 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* $x \in \mathit{Inv}(p)$*. The set of all markings over* N *is denoted by* $\mathcal{M}(N)$*.*

We shall sometimes use the notation (p, x) to refer to a token in the place p of age $x \in \mathbb{R}_{\geq 0}$. Likewise, we shall sometimes write $M = \{(p_1, x_1), (p_2, x_2), \ldots, (p_n, x_n)\}$ for a multiset representing a marking M with n tokens located in the places p_i and with age x_i for $1 \leq i \leq n$.

A *marked* TAPN is a pair (N, M_0) where N is a TAPN and M_0 is an initial marking on N where all tokens have the age 0.

Definition 3 (Enabledness). *Let $N = (P, T, IA, OA, \mathit{Transport}, \mathit{Inhib}, \mathit{Inv})$ be a TAPN. We say that a transition $t \in T$ is* enabled *in a marking M by tokens $In = \{(p, x_p) \mid p \in {}^{\bullet}t\} \subseteq M$ and $Out = \{(p', x_{p'}) \mid p' \in t^{\bullet}\}$ if*

– *for all input arcs except the inhibitor arcs there is a token in the input place with an age satisfying the age guard of the arc, i.e.*

$$\forall (p, I, t) \in IA \,.\, \neg \mathit{Inhib}((p, I, t)) \Rightarrow x_p \in I$$

– *for all inhibitor arcs there is no token in the input place of the arc with an age satisfying the age guard of the arcs respectively, i.e.*

$$\forall (p, I, t) \in IA \,.\, \mathit{Inhib}((p, I, t)) \Rightarrow \neg \exists x \in M(p) \,.\, x \in I$$

– *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.*

$$\forall (p, I, t) \in IA \,.\, \forall (t, p') \in OA \,.\, \mathit{Transport}((p, I, t), (t, p')) \Rightarrow$$
$$(x_p = x_{p'}) \ \wedge \ (x_p \in \mathit{Inv}(p'))$$

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

$$\forall (t, p') \in OA \,.\big(\neg(\exists \alpha \in IA. \mathit{Transport}(\alpha, (t, p'))) \ \Rightarrow x_{p'} = 0\big) \ .$$

Definition 4 (Firing Rule). *Let $N = (P, T, IA, OA, \mathit{Transport}, \mathit{Inhib}, \mathit{Inv})$ be a TAPN, M a marking on N and $t \in T$ a transition . If t is enabled in the marking M by tokens In and Out then it can* fire *and produce a marking M' defined as*

$$M' = (M \setminus In) \cup Out$$

where $\setminus$ and $\cup$ are operations on multisets.

Definition 5 (Time Delay). *Let $N = (P, T, IA, OA, \mathit{Transport}, \mathit{Inhib}, \mathit{Inv})$ be a TAPN and M a marking on N. A* time delay *$d \in \mathbb{R}_{\geq 0}$ is allowed in M if $(x + d) \in \mathit{Inv}(p)$ for all $p \in P$ and all $x \in M(p)$, i.e. by delaying d time units no token violates any of the age invariants. By delaying d time units in M we reach a marking M' defined as*

$$M'(p) = \{x + d \mid x \in M(p)\}$$

for all $p \in P$.

A given TAPN N now defines a timed transition system $(\mathcal{M}(N), \longrightarrow)$ where states are markings of N and for two markings M and M' we have $M \longrightarrow M'$ if by firing some transition in M we can reach the marking M' and $M \xrightarrow{d} M'$ if by delaying d time units in M we reach the marking M'. We say that a marking M' is reachable from marking M if $M \Longrightarrow^* M'$.

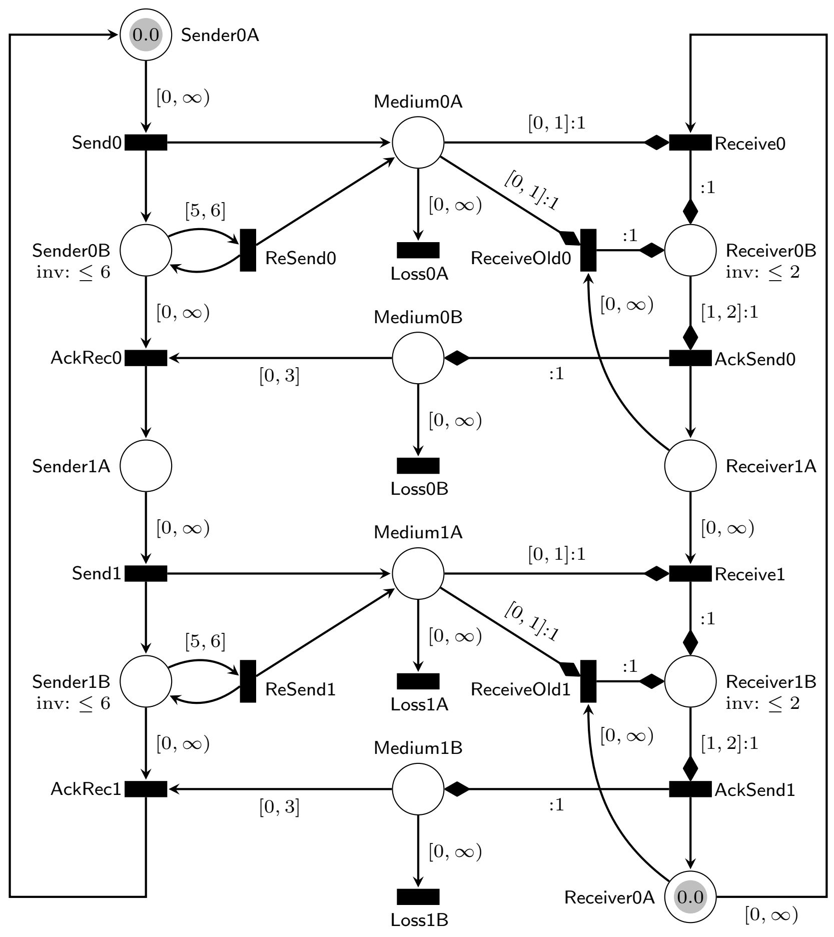


Fig. 2: A TAPN model of the alternating bit protocol

4 A Small Case Study: Alternating Bit Protocol

In this section we shall discuss a small case study of the well-known Alternating Bit Protocol (ABP) [9] and show its modelling by timed-arc Petri nets. The purpose of the protocol is to ensure a safe communication between a sender and a receiver over an unreliable medium. To achieve this, messages are labelled with a control bit in order to compensate (via message retransmission) for the possibility of losing messages in transfer. In order to avoid a confusion between new and old messages, each message is moreover time-stamped and after its expiration it is ignored.

Figure 2 shows a TAPN model of the protocol. The model contains four places associated to the sender (on the left side) and four places associated to the receiver (on the right side). The sender and receiver can communicate over a lossy medium represented by the four places in the middle.

Initially, the only enabled transition is Send0 which moves the sender from the place Sender0A to Sender0B and at the same time places a message (token) with the appended bit 0 into the place Medium0A. At any time the message can be lost by firing the transition Loss0A. Now the receiver can read the message and move it to the place Receiver0B by firing the transition Receive0, followed by firing AckSend0 and placing a token (acknowledgment) in the place Medium0B. There is at least a one time unit delay before the acknowledgment is sent. The acknowledgment can now be read by the sender using the transition AckRec0. Notice that the path of the token from place Medium0A to Medium0B consists of transport arcs and hence the time-stamp of the message is preserved. The sender accepts only acknowledgments that are no older than three time units since the message was sent. As the medium is lossy, the communication may fail and it may be necessary to retransmit the message by firing the transition ReSend0, which must happen any time between five to six time units since the last time the message was sent. Similarly, the receiver may retransmit the acknowledgment by firing the transitions ReceiveOld0 and AckSend0. If the first phase with the appended bit 0 succeeded, the protocol continues in a symmetric way with the next message that gets appended the bit 1.

Having the formal model of alternating bit protocol in place, we can now start analysing its behaviour. One possible analysis technique is the simulation of transition firings that can reveal possible flaws in the design, however, it cannot be used to argue about the correctness of the protocol. We will postpone the actual definition of the correctness requirement of the protocol to Section 6 once an appropriate logic for the formulation of this property is defined. In the meantime we can observe that the net can exhibit a behaviour in which the places representing the medium become unbounded (there is no a priori given constant that bounds the number of tokens in these places). This can be seen by the fact that e.g. the transition ReSend0 can be repeatedly fired and, as the net is not forced to ever perform the transition Loss0A, more and more tokens will accumulate in the place Medium0A. In the section to follow, we will show that automatic verification of unbounded nets with invariants is not possible, as the model has the full Turing power.

A possible solution to this problem (which is though specific to our concrete model) is to introduce age invariants to all places representing the medium which will disallow tokens older than two time units. This will enforce urgency on the transitions that lose messages and it can be proved (or automatically verified) that the net becomes bounded after such an addition, while the interesting behaviour of the protocol does not change.

Another approach that works in general is to consider an under-approximation of the net behaviour where we give a limit on the maximum number of new tokens that the net can produce and explore the behaviour of the net only up to that many tokens. An experiment using this approach is described in Section 7.

A similarly looking model of the alternating bit protocol was given also for time Petri nets (see e.g. [11]) where clocks are associated to each transition of the net. Unlike our TAPN model, TPN do not allow time-stamps on tokens and messages are instead automatically discarded after one time unit. Hence the behaviour of the TPN is less general and does not allow us to model (at least not in a straightforward way) all the features available in TAPN.

5 Overview of (Un)Decidability and Complexity Results

In this section we shall discuss results about (un)decidability and complexity questions of the classical Petri net problems like reachability, coverability and boundedness in the TAPN context.

We start with the problem of *reachability*: given a marked net (N, M_0) and a marking M, is M reachable from M_0? In spite of the fact that reachability is decidable for untimed Petri nets [37], it is undecidable for timed-arc Petri nets [47], even for nets without any transport/inhibitor arcs and age invariants. The result can be further extended to the case where tokens in different places are not required to age synchronously [41].

We shall now recall the idea of the undecidability result by Ruiz et al. [47] as it is easy to explain and illustrates the power of tokens with age. For showing the undecidability of many Petri net problems the halting problem for Minsky two counter machine is often exploited. The proof from [47] is no exception.

A *Minsky machine* with two nonnegative counters c_1 and c_2 is a sequence of labelled instructions

$$1 : \mathsf{inst}_1;\ 2 : \mathsf{inst}_2;\ \ldots, n : \mathsf{inst}_n$$

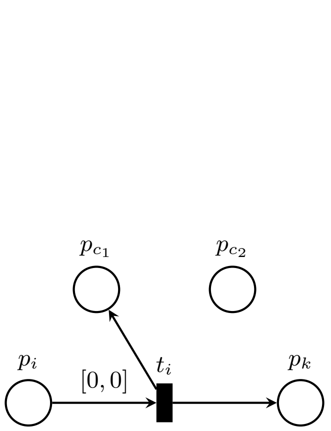
where $\mathsf{inst}_n = \mathsf{HALT}$ and each inst_i, $1 \leq i < n$, is of one of the following forms

- (Inc) i: c_j`++; goto` k
- (Dec) i: `if` c_j`=0 then goto` k `else (`c_j`--; goto` ℓ`)`

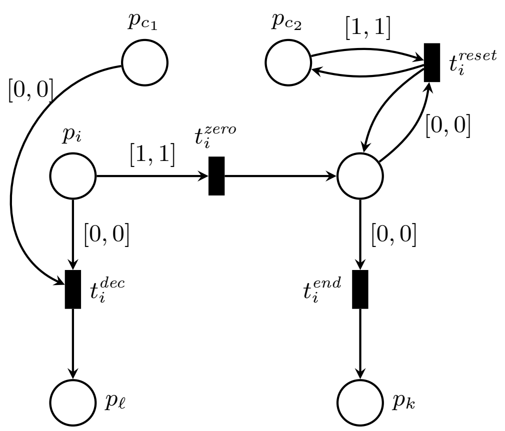
for $j \in \{1, 2\}$ and $1 \leq k, \ell \leq n$.

Instructions of type (Inc) are called *increment* instructions and of type (Dec) are called *test and decrement* instructions. A configuration is a triple (i, v_1, v_2) where i is the current instruction and v_1 and v_2 are the values of the counters c_1 and c_2, respectively. A computation step between configurations is defined in the natural way. If starting from the initial configuration $(1, 0, 0)$ the machine reaches the instruction HALT then we say it *halts*, otherwise it *loops*. It is well known that the problem whether a given Minsky machine halts is undecidable [40]. This is the case even for the question whether it halts with both counters empty (as they can be easily emptied before the halting instruction is reached).

The main idea of simulating a Minsky machine by a Petri net is to create two places called p_{c_1} and p_{c_2} such that the number of tokens in these places represents the value of the counters c_1 and c_2, respectively. Also, for every instruction label i, $1 \leq i \leq n$, we create a new place called p_i in the net. During the behaviour



(a) i: `c`$_1$`++; goto` k



(b) i: `if` c_1`=0 then goto` k `else (`c_1`--; goto` ℓ`)`

Fig. 3: Simulation of (Inc) and (Dec) instructions by basic TAPN

of the net the sum of the tokens in $p_1, \ldots, p_n$ will be invariantly equal to one. The presence of a token at p_i represents the fact that the next instruction to be executed is the one with label i.

Given a Minsky machine with two counters, we shall now construct a basic TAPN such that, given an initial marking with just one token in p_1, the final marking where there is exactly one token of age 0 in place p_n is reachable if and only if the given Minsky machine halts.

The instruction of type (Inc) is easy to simulate as depicted in Figure 3a for the increment of c_1; a symmetric construction is used for the increment of c_2. The interval $[0, 0]$ disallows any time delay before the transition t_i is fired. After the firing, the control is given to the instruction k and the counter c_1 is incremented by one. Note that there is no invariant in p_i so we are also allowed to delay here but in this case the whole net will get stuck and it will thus not be possible to place a token at the place p_n.

For any instruction of type (Dec) we add the places and transitions as depicted in Figure 3b (again the test on counter c_2 is completely symmetric). The modelling of the decrement branch via the transition t_i^{dec} is straightforward. The difficult part is the simulation of the jump to label k when the counter c_1 is empty. As Petri nets, unless equipped with inhibitor arcs, do not allow to test for zero number of tokens in a place, we need to introduce a few more transitions that will detect a *cheating*, i.e. when the transition t_i^{zero} is taken while there are some tokens in p_{c_1}. Notice that the transition t_i^{zero} can be fired only after a delay of one time unit, hence all tokens in both p_{c_1} and p_{c_2} will also grow older by one time unit. Now the transition t_i^{reset} will allow to reset the ages of all tokens in p_{c_2} to 0, however, any potential tokens in place p_{c_1} will remain of age 1 (if we cheated). The simulation then continues by firing the transition t_i^{end} which gives the control to the instruction k.

Clearly, if the given Minsky machine halts with both counters empty, we can faithfully simulate its computation in the Petri net such that the place p_n will be eventually marked and all other places will be empty.

On the other hand, if the Minsky machine loops then we can either faithfully simulate it in the net but then the final marking with one token in p_n will never be reached, or we can cheat but as a result the net will contain tokens which are too old (also called dead tokens) in either p_{c_1} or p_{c_2} that cannot be removed, and hence the final marking will not be reachable either.

Theorem 1 ([47]). *Reachability is undecidable for the basic timed-arc Petri net model (with only ordinary arcs and no age invariants).*

On the other hand, coverability, boundedness and other problems remain decidable for the basic TAPN [4, 6, 48] model, which is also known to offer 'weak' expressiveness, in the sense that basic TAPN cannot simulate Turing machines [14].

The *coverability problem* asks, given an initial marking M_0 and a final marking M, is there a marking M' reachable from M_0 such that $M(p) \subseteq M'(p)$ for all places p?

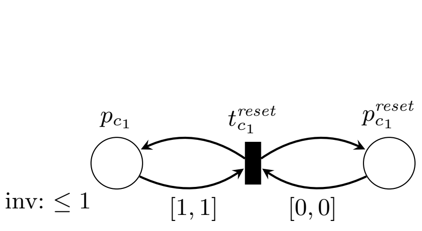
The *boundedness problem* asks, given an initial marking M_0, is there a constant k such that the total number of tokens in any reachable marking from M_0 is less than or equal to k? If this is the case, the net is called k-bounded.

It is known that coverability remains decidable for the basic TAPN extended with read arcs [17] where a read arc is a special case of a pair of two transport arcs that return the consumed token to the same place (and hence do not change its age). These results hold due to the monotonicity property (adding more tokens to the net does not restrict the possible executions) and the application of well-quasi-ordering (for a general introduction see [25]) resp. better-quasi-ordering [3] techniques.

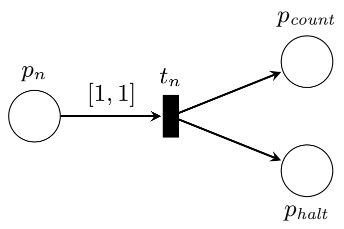
One of the major weaknesses of the basic TAPN is the lack of the possibility to model urgent behaviour. On the other hand, when allowing age invariants, both coverability and boundedness become undecidable as shown in [30] and demonstrated in what follows.

The basic idea is similar as in the previous reduction and illustrations are depicted in Figure 4. We have two places p_{c_1} and p_{c_2} representing the counters plus we add one more place called p_{count} which records the number of already executed instructions and will be used for the undecidability of boundedness. The counters are each equipped with a place called $p_{c_j}^{reset}$ such that a presence of a token in this place will allow us to reset the age of all tokens of age 1 in p_{c_j}.

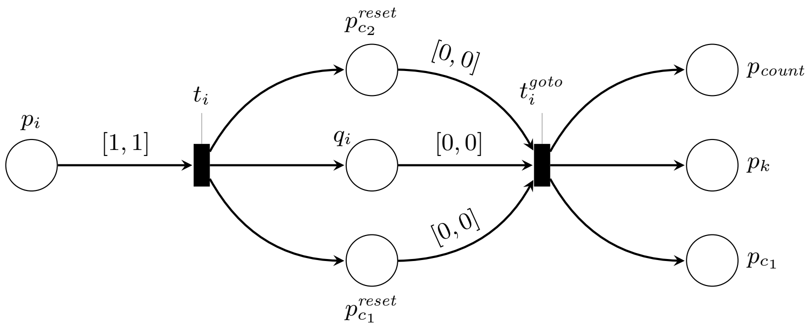
The simulation of the increment instruction in Figure 4c starts by delaying one time unit. Now all tokens in the counters become of age 1 but can subsequently be reset to 0 due to the presence of the tokens in $p_{c_1}^{reset}$ and $p_{c_2}^{reset}$. By performing t_i^{goto} the simulating finishes, increases the number of tokens in p_{count} by one, gives the control to the instruction k and adds 1 to the counter c_1.



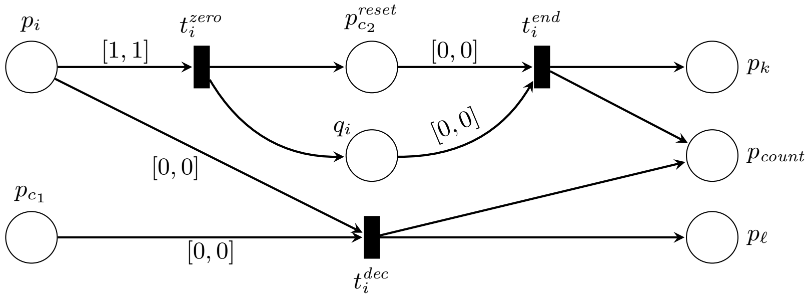
(a) Simulation of the counter c_1



(b) Simulation of instruction halt



(c) i: c_1`++; goto` k



(d) i: `if` c_1`=0 then goto` k `else` (c_1`--; goto` ℓ).

Fig. 4: Simulation of a Minsky machine by a TAPN with invariants

The decrement instruction, depicted in Figure 4d, can fire the transition t_i^{dec}, provided that there is a token of age 0 in p_{c_1}, increase the number of counted steps and give the control to the instruction ℓ. It can also delay one time unit and perform the transition t_i^{zero} which will allow us to reset the ages of tokens in p_{c_2} and after firing t_i^{end} add one token to p_{count} and continue with the simulation of the instruction k. The point is that if we were cheating in the simulation and fired the transition t_i^{zero} with a nonempty counter c_1, tokens of age 1 will necessarily appear in the place p_{c_1}. Notice that the simulation of most instructions (in particular of the halt instruction) must start with a time delay,

	Reachability	**Coverability**	**Boundedness**
basic TAPN	× [47]	✓ [4]	✓ [6]
basic TAPN plus transport arcs	× [47]	$\checkmark^?$	$\checkmark^?$
basic TAPN plus age invariants	× [47]	× [30]	× [30]
basic TAPN plus inhibitor arcs	× [27]	× [27]	× [27]

Table 1: Overview of (un)decidability results for TAPN

however, if in some of the counters there were tokens of age 1, no time delay is possible due to the age invariants ≤ 1 in p_{c_1} and p_{c_2}. Hence the place p_{halt} can be marked if and only if the net did not cheat and this gives the undecidability of coverability.

The same construction also serves as a reduction showing undecidability of boundedness. Assume that the given Minsky machine halts in k steps. This means that if the net faithfully simulates its behaviour, it will terminate with a token in p_{halt} and at most k tokens in any of the two counter places and p_{count}. If the net cheated at some point, most of the instructions will be disabled as discussed above, except for the firing of t_i^{dec}, which can however only decrease the number of tokens in the places. The net is hence bounded. On the other hand, if the Minsky machine loops, the net can faithfully simulate this infinite behaviour and the place p_{count} will be unbounded. Hence the undecidability of boundedness for basic TAPN with invariants is established too.

Theorem 2 ([30]). *Coverability and boundedness are undecidable for basic TAPN with invariants.*

A summary of the results is provided in Table 1. The decidability of coverability and boundedness for TAPN with transport arcs is marked with a question mark as it is only a claim, though the proofs from [17] for read arcs seem easy to extend to TAPN with transport arcs too.

In applications, the fact that coverability is decidable for the basic TAPN model can be useful as demonstrated in [4] where the authors verified a parameterized version of Fischer's protocol [35] using their prototype implementation of the coverability algorithm.

Most often though, we may like to use the additional features like age invariants and inhibitor arcs to facilitate the modelling process. While all interesting problems become quickly undecidable for such models, we may still verify a number of interesting properties by restricting ourselves to bounded nets where the maximum number of tokens in the net is given as a constant. Recent work shows that bounded TAPN and 1-safe (at most one token in any place) nets offer a similar expressive power as networks of timed automata, even though the models are rather different. Sifakis and Yovine [52] provided a translation of 1-safe timed-arc Petri nets into timed automata which preserves strong timed bisimilarity but their translation causes an exponential blow up in the size. Srba established in [53] a strong relationship (up to isomorphism of timed transition systems) between networks of timed automata and a superclass of 1-safe TAPN extended with read arcs. For reachability questions the reductions in [53] work in polynomial time. Recently Bouyer et al. [17] presented a reduction from bounded TAPN (with read-arcs) to 1-safe TAPN (with read-arcs), which preserves timed language equivalence. Hence PSPACE-completeness of reachability on 1-safe and bounded TAPN was established [17, 53].

Nevertheless the translations described in these papers are inefficient from the practical point of view as they either cause an exponential blow-up in the size or create a new parallel component with a fresh local clock for *each place* in the net. In connection with the development of the tool TAPAAL [1] for modelling, simulation and verification of extended timed-arc Petri nets, more efficient translations were investigated [21, 22].

Recently, in [32] we identified a general class of translations that preserve Timed Computation Tree Logic (TCTL), a logic suitable for practical specification of many useful temporal properties (see e.g. [44]). In the next two sections we shall present the framework and give an example of an efficient translation from TAPN to UPPAAL networks of timed automata [2].

6 Timed Computation Tree Logic

In this section we introduce the Timed Computation Tree Logic (TCTL). Unlike much work on TCTL where only infinite alternating runs are considered [44] or the details are simply not discussed [16, 23], we consider also finite maximal runs that appear in the presence of stuck computations or time invariants (strict or nonstrict) and treat the semantics in its full generality as used in most of the verification tools nowadays. This fact is particularly important for the verification of liveness properties.

Before we define the syntax and semantics of TCTL, we extend the notion of timed transition systems (TTS) as defined in Section 3 with propositions. A *timed transition system with propositions* is a quadruple $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ where $(S, \longrightarrow)$ is a TTS, $\mathcal{AP}$ is a set of atomic propositions, and $\mu : S \to 2^{\mathcal{AP}}$ is a function assigning sets of true atomic propositions to states.

For a TAPN $N = (P, T, \mathit{IA}, \mathit{OA}, \mathit{Transport}, \mathit{Inhib}, \mathit{Inv})$ the set of atomic propositions $\mathcal{AP}$ and the labeling function μ can be defined as

$$\mathcal{AP} \overset{\mathit{def}}{=} \{(p \bowtie n) \mid p \in P, n \in \mathbb{N}_0 \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\}$$

and for a marking M we have

$$\mu(M) \overset{\mathit{def}}{=} \{(p \bowtie n) \mid |M(p)| \bowtie n \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\} \ .$$

The intuition is that a proposition $(p \bowtie n)$ is true in a marking M iff the number of tokens in the place p satisfies the given relation with respect to n.

A *run* $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} \ldots$ in a TTS is a (finite or infinite) alternating sequence of time delays and discrete actions.

We shall now introduce the syntax and semantics of TCTL. The presentation is inspired by [44]. Let $\mathcal{AP}$ be a set of atomic propositions. The set of TCTL formulae $\Phi(\mathcal{AP})$ over $\mathcal{AP}$ is given by

$$\varphi ::= \wp \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid E(\varphi_1\, U_I\, \varphi_2) \mid A(\varphi_1\, U_I\, \varphi_2) \mid E(\varphi_1\, R_I\, \varphi_2) \mid A(\varphi_1\, R_I\, \varphi_2)$$

where $\wp \in \mathcal{AP}$ ranges over atomic propositions and $I \in \mathcal{I}$ ranges over time intervals. Formulae without any occurrence of the operators $A(\varphi_1\, U_I\, \varphi_2)$ and $E(\varphi_1\, R_I\, \varphi_2)$ form the *safety fragment* of TCTL.

The intuition of the until and release TCTL operators (formalized later on) is as follows:

- $A(\varphi_1\, U_I\, \varphi_2)$ is true if on all maximal runs φ_2 eventually holds within the interval I, and until it does, φ_1 continuously holds;
- $E(\varphi_1\, U_I\, \varphi_2)$ is true if there exists a maximal run such that φ_2 eventually holds within the interval I, and until it does, φ_1 continuously holds;
- $A(\varphi_1\, R_I\, \varphi_2)$ is true if on all maximal runs either φ_2 always holds within the interval I or φ_1 occurred previously;
- $E(\varphi_1\, R_I\, \varphi_2)$ is true if there exists a maximal run such that either φ_2 always holds within the interval I or φ_1 occurred previously.

In the semantics, we handle maximal runs in their full generality. Hence we have to consider all possibilities in which a run can be "stuck". In this case, we annotate the last transition of such a run with one of the three special ending symbols (denoted δ in the definition below).

A *maximal run* ρ is either

(i) an infinite alternating sequence of the form $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots$, or

(ii) a finite alternating sequence of the form $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow \ldots \longrightarrow s_n \xrightarrow{\delta}$ where $\delta \in \{\infty, d_n^{\leq}, d_n^{<}\}$ for some $d_n \in \mathbb{R}_{\geq 0}$ s.t.
  - if $\delta = \infty$ then $s_n \xrightarrow{d} s_n[d]$ for all $d \in \mathbb{R}_{\geq 0}$,
  - if $\delta = d_n^{\leq}$ then $s_n \not\xrightarrow{d}$ for all $d > d_n$ and $s_n \xrightarrow{d_n} s_n[d_n]$ s.t. $s_n[d_n] \not\longrightarrow$, and
  - if $\delta = d_n^{<}$ then $s_n \not\xrightarrow{d}$ for all $d \geq d_n$, and there exists d_s, $0 \leq d_s < d_n$, such that for all d, $d_s \leq d < d_n$, we have $s_n \xrightarrow{d} s_n[d]$ and $s_n[d] \not\longrightarrow$.

By $\mathit{MaxRuns}(T, s)$ we denote the set of maximal runs in a TTS T starting at s.

Intuitively, the three conditions in case (ii) describe all possible ways in which a finite run can terminate. First, a run can end in a state where time diverges. The other two cases define a run which ends in a state from which no discrete transition is allowed after some time delay, but time cannot diverge either (typically caused by the presence of invariants in the model). These cases differ in whether the bound on the maximal time delay can be reached or not.

Figure 5 illustrates a part of a maximal run $\rho = s_0 \xrightarrow{1} s_0[1] \longrightarrow s_1 \xrightarrow{2.5} s_1[2.5] \longrightarrow s_2 \xrightarrow{2} s_2[2] \longrightarrow s_3 \xrightarrow{1.3} s_3[1.3] \longrightarrow s_4 \longrightarrow \ldots$. Note that actions take

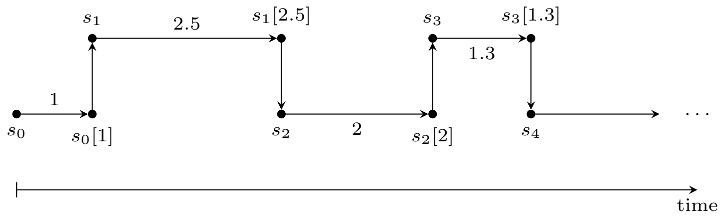


Fig. 5: Illustration of a concrete run

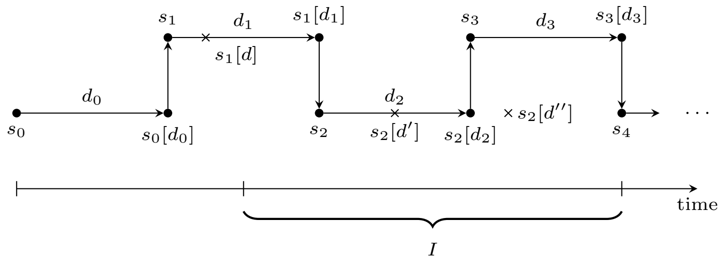


Fig. 6: Illustration of a run

zero time units and that, although not shown in this example, time delays can be zero so it is possible to do multiple actions in succession without any time progression in between. Further, there is no special meaning as to whether the arrow for an action goes up or down, this is simply to keep the figure small.

Let us now introduce some notation for a given maximal run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} \ldots$. First, $r(i,d)$ denotes the total time elapsed from the beginning of the run up to some delay $d \in \mathbb{R}_{\geq 0}$ after the i'th discrete transition. Formally, $r(i,d) = \left(\sum_{j=0}^{i-1} d_j\right) + d$. Second, we define a predicate $\mathit{valid}_\rho : \mathbb{N}_0 \times \mathbb{R}_{\geq 0} \times \mathcal{I} \to \{\mathit{true}, \mathit{false}\}$ such that $\mathit{valid}_\rho(i,d,I)$ checks whether the total time for reaching the state $s_i[d]$ in ρ belongs to the time interval I, formally

$$\mathit{valid}_\rho(i,d,I) = \begin{cases} d \leq d_i \wedge r(i,d) \in I & \text{if } d_i \in \mathbb{R}_{\geq 0} \\ r(i,d) \in I & \text{if } d_i = \infty \\ d \leq d_n \wedge r(i,d) \in I & \text{if } d_i = d_n^{\leq} \\ d < d_n \wedge r(i,d) \in I & \text{if } d_i = d_n^{<} \ . \end{cases}$$

Let us now give some example of the application of the $\mathit{valid}_\rho(i,d,I)$ function. Figure 6 illustrates a run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots$ and three points (marked with $\times$). We see that $\mathit{valid}_\rho(1,d,I)$ is false because $s_1[d]$ lies outside the interval I. Similarly, $\mathit{valid}_\rho(2,d'',I)$ is false because $s_2[d'']$ is not a part of the run (since $d'' > d_2$). Finally, $\mathit{valid}_\rho(2,d',I)$ is true because $s_2[d']$ is a part of the run and within I.

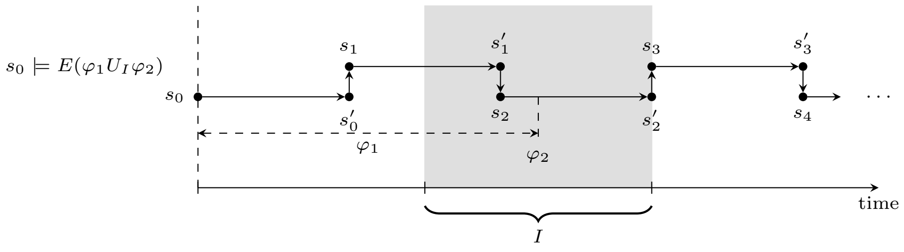


Fig. 7: Illustration of a run satisfying an until formula.

Next, we define a function $history_\rho : \mathbb{N}_0 \times \mathbb{R}_{\geq 0} \to 2^{\mathbb{N}_0 \times \mathbb{R}_{\geq 0}}$ s.t. $history_\rho(i, d)$ returns the set of pairs (j, d') that constitute all states $s_j[d']$ in ρ preceding $s_i[d]$, formally $history_\rho(i, d) = \{(j, d') \mid 0 \leq j < i \wedge 0 \leq d' \leq d_j\} \cup \{(i, d') \mid 0 \leq d' < d\}$.

Now we can define the satisfaction relation $s \models \varphi$ for a state $s \in S$ in a TTS with propositions $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ and a TCTL formula φ.

$$
\begin{array}{ll}
s \models \wp & \text{iff } \wp \in \mu(s) \\
s \models \neg\varphi & \text{iff } s \not\models \varphi \\
s \models \varphi_1 \wedge \varphi_2 & \text{iff } s \models \varphi_1 \text{ and } s \models \varphi_2 \\
s \models E(\varphi_1 \, U_I \, \varphi_2) & \text{iff } \exists \rho \in \mathit{MaxRuns}(T, s) \, . \\
& \quad \exists i \geq 0 \, . \, \exists d \in \mathbb{R}_{\geq 0} \, . \, [\mathit{valid}_\rho(i, d, I) \wedge s_i[d] \models \varphi_2 \wedge \\
& \quad \forall (j, d') \in \mathit{history}_\rho(i, d) \, . \, s_j[d'] \models \varphi_1] \\
s \models E(\varphi_1 \, R_I \, \varphi_2) & \text{iff } \exists \rho \in \mathit{MaxRuns}(T, s) \, . \\
& \quad \forall i \geq 0 \, . \, \forall d \in \mathbb{R}_{\geq 0} \, . \, \mathit{valid}_\rho(i, d, I) \Rightarrow \\
& \quad \left[s_i[d] \models \varphi_2 \vee \exists (j, d') \in \mathit{history}_\rho(i, d) \, . \, s_j[d'] \models \varphi_1 \right]
\end{array}
$$

The operators $A(\varphi_1 \, U_I \, \varphi_2)$ and $A(\varphi_1 \, R_I \, \varphi_2)$ are defined analogously by replacing the quantification $\exists \rho \in \mathit{MaxRuns}(T, s)$ with $\forall \rho \in \mathit{MaxRuns}(T, s)$.

Figure 7 illustrates the satisfaction of the until formula and Figure 8 illustrates the release formula. In particular, notice that there are four possible ways for a release formula to be satisfied. First, φ_1 may have occurred in the past (outside the interval), which releases φ_2, effectively ensuring that φ_2 need not hold in the interval I at all. Second, φ_2 may not be released, which means that it must hold continuously within the entire interval I. Third, φ_2 can hold continuously in the interval I, until some point in the interval where $\varphi_1 \wedge \varphi_2$ holds, thereby releasing φ_2. Finally, φ_2 can hold continuously in the interval I until the run deadlocks.

As expected, the until and release operators are dual.

Lemma 1 ([32]). *We have* $s \models A(\varphi_1 \, R_I \, \varphi_2)$ *iff* $s \models \neg E(\neg\varphi_1 \, U_I \, \neg\varphi_2)$, *and* $s \models A(\varphi_1 \, U_I \, \varphi_2)$ *iff* $s \models \neg E(\neg\varphi_1 \, R_I \, \neg\varphi_2)$.

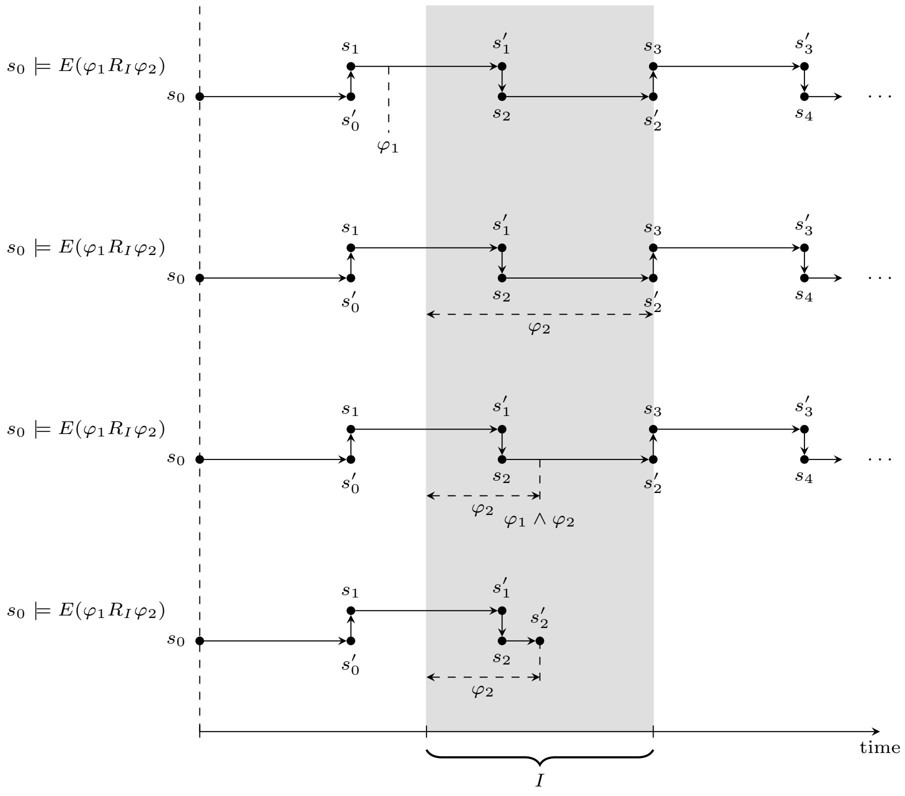


Fig. 8: Illustration of runs satisfying a release formula.

Example 1. Consider again the TAPN model of alternating bit protocol from Section 4. We can express the correctness of the protocol as the property that the sender and receiver never get out of synchrony. This property is violated if the sender is about to send a message with the bit 0 but the receiver is either in the state Receiver0B or Receiver1A, in other words when the receiver is sending or resending an acknowledgment for the bit 0. Such situation should not happen, and symmetrically for the second part of the protocol where a message with the bit 1 is about to be sent. We can express the violation of synchrony by the following TCTL formula: $E(\mathit{true}\, U_{[0,\inf)}\, (\mathsf{Sender0A} = 1 \wedge (\mathsf{Receiver0B} = 1 \vee \mathsf{Receiver1A} = 1)) \vee (\mathsf{Sender1A} = 1 \wedge (\mathsf{Receiver1B} = 1 \vee \mathsf{Receiver0A} = 1)))$. The time interval in the until operator is set to $[0, \inf)$ as the correct protocol behaviour should not be violated at any point of its execution. In Section 7 we discuss automatic tool-supported verification of this property.

Another example of a property can require that during the first 20 time units of the protocol execution there are never more than 5 acknowledgment messages in transfer. This can be expressed by the TCTL formula

$A(\mathit{false}\, R_{[0,20]}\, (\mathsf{Medium0B} \leq 5 \wedge \mathsf{Medium1B} \leq 5))$ and it is satisfied in the initial marking of the alternating bit protocol.

Finally, we may also ask whether the sender and the receiver eventually finish the transmission of the message with bit 0 and proceed to a message with bit 1. However, the TCTL formula $A(\mathit{true}\, U_{[0,\infty)}\, (\mathsf{Sender1A} = 1 \wedge \mathsf{Receiver1A} = 1))$ expressing this property is false due to several reasons. First of all, in the initial marking the sender is not forced to initiate the sending of the first message and time can elapse for ever. This can be fixed by adding age invariants at all sender and receiver places in order to enforce urgency. However, as the medium is lossy, there is another maximal run where the retransmitted message gets repeatedly lost and such run also violates our formula.

7 Translations Preserving TCTL Model Checking

In this section, we shall present a general framework for arguing when a simulation of one time dependent system by another preserves satisfiability of TCTL formulae. We define the notion of one-by-many correspondence, a relation between two TTSs A and B, such that if A is in one-by-many correspondence with B then every transition in A can be simulated by a sequence of transitions in B. Further, every TCTL formula φ can be algorithmically translated into a formulate $tr(\varphi)$ s.t. $A \models \varphi$ iff $B \models tr(\varphi)$. In the rest of this section, we shall use A and B to refer to the original and the translated system, respectively. The text of the next subsection is to a large extend based on [32] where the reader can find a more detailed exposition and proofs.

7.1 One-By-Many Correspondence

As the system B is simulating a single transition of A by a sequence of transitions, the systems A and B are comparable only in the states before and after this sequence was performed. We say that B is *stable* in such states and introduce a fresh atomic proposition called *stable* to explicitly identify this situation. We now define three conditions that B should possess in order to apply to our framework. A TTS $(S, \rightarrow, \mathcal{AP}, \mu)$ s.t. $\mathit{stable} \in \mathcal{AP}$ is

- *delay-implies-stable* if for any $s \in S$, it holds that $s \xrightarrow{d}$ for some $d > 0$ implies $s \models \mathit{stable}$,
- *delay-preserves-stable* if for any $s \in S$ such that $s \models \mathit{stable}$, if $s \xrightarrow{d} s[d]$ then $s[d] \models \mathit{stable}$ for all $d \in \mathbb{R}_{\geq 0}$, and
- *eventually-stable* if for any $s_0 \in S$ such that $s_0 \models \mathit{stable}$ and for any infinite sequence of discrete transitions $\rho = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow s_4 \longrightarrow \ldots$ or any finite nonempty sequence of discrete transitions $\rho = s_0 \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_n \not\longrightarrow$ there exists an index $i \geq 1$ such that $s_i \models \mathit{stable}$.

We write $s \rightsquigarrow s'$ if there is a sequence $s = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_n = s'$ s.t. $s \models \mathit{stable}$, $s' \models \mathit{stable}$, and $s_j \not\models \mathit{stable}$ for $1 \leq j \leq n-1$.

Definition 6. *Let $A = (S, \rightarrow_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \rightarrow_B, \mathcal{AP}_B, \mu_B)$ be two TTSs s.t. $stable \in \mathcal{AP}_B$ and B is a* delay-implies-stable *and* delay-preserves-stable *TTS. A relation $\mathcal{R} \subseteq S \times T$ is a* one-by-many correspondence *if there exists a function $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$ such that whenever $s\,\mathcal{R}\,t$ then*

1. *$t \models stable$,*
2. *$s \models \wp$ iff $t \models tr_p(\wp)$ for all $\wp \in \mathcal{AP}_A$,*
3. *if $s \longrightarrow s'$ then $t \rightsquigarrow t'$ and $s'\,\mathcal{R}\,t'$,*
4. *if $s \xrightarrow{d} s[d]$ then $t \xrightarrow{d} t[d]$ and $s[d]\,\mathcal{R}\,t[d]$ for all $d \in \mathbb{R}_{\geq 0}$,*
5. *if $t \rightsquigarrow t'$ then $s \longrightarrow s'$ and $s'\,\mathcal{R}\,t'$, and*
6. *if $t \xrightarrow{d} t[d]$ then $s \xrightarrow{d} s[d]$ and $s[d]\,\mathcal{R}\,t[d]$ for all $d \in \mathbb{R}_{\geq 0}$.*

If B is moreover an eventually-stable *TTS, then we say that $\mathcal{R}$ is a* complete one-by-many correspondence. *We write $s \mathrel{\overset{\supset}{=}} t$ (resp. $s \mathrel{\overset{\supset}{=}}_c t$) if there exists a relation $\mathcal{R}$ which is a one-by-many correspondence (resp. a complete one-by-many correspondence) such that $s\,\mathcal{R}\,t$.*

Now we translate TCTL formulae. Let $\mathcal{AP}_A$ and $\mathcal{AP}_B$ be sets of atomic propositions such that $stable \in \mathcal{AP}_B$ and let $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$ be a function translating atomic propositions. We define $tr : \Phi(\mathcal{AP}_A) \rightarrow \Phi(\mathcal{AP}_B)$ as follows.

$$\begin{aligned}
tr(\wp) &= tr_p(\wp)\\
tr(\neg\varphi_1) &= \neg tr(\varphi_1)\\
tr(\varphi_1 \wedge \varphi_2) &= tr(\varphi_1) \wedge tr(\varphi_2)\\
tr(E(\varphi_1\, U_I\, \varphi_2)) &= E((tr(\varphi_1) \vee \neg stable)\, U_I\, (tr(\varphi_2) \wedge stable))\\
tr(A(\varphi_1\, U_I\, \varphi_2)) &= A((tr(\varphi_1) \vee \neg stable)\, U_I\, (tr(\varphi_2) \wedge stable))\\
tr(E(\varphi_1\, R_I\, \varphi_2)) &= E((tr(\varphi_1) \wedge stable)\, R_I\, (tr(\varphi_2) \vee \neg stable))\\
tr(A(\varphi_1\, R_I\, \varphi_2)) &= A((tr(\varphi_1) \wedge stable)\, R_I\, (tr(\varphi_2) \vee \neg stable))
\end{aligned}$$

We are now ready to state the main result (see [33] for its full proof).

Theorem 3. *Let $A = (S, \rightarrow_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \rightarrow_B, \mathcal{AP}_B, \mu_B)$ be two TTSs such that $stable \in \mathcal{AP}_B$ and let $s_0 \in S$ and $t_0 \in T$. If $s_0 \mathrel{\overset{\supset}{=}}_c t_0$ then for any TCTL formula φ we have $s_0 \models \varphi$ if and only if $t_0 \models tr(\varphi)$. If $s_0 \mathrel{\overset{\supset}{=}} t_0$ then the claim holds only for any formula φ from the safety fragment of TCTL.*

We finish this subsection by recalling the steps needed in order to apply the framework to a particular translation between two time-dependent systems. Assume that we designed an algorithm that for a given system A constructs a system B together with the notion of stable states in the system B.

1. Show that B is a *delay-implies-stable* and *delay-preserves-stable* TTS (and optionally an *eventually-stable* TTS).
2. Define a proposition translation function $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$.
3. Define a relation $\mathcal{R}$ and show that it fulfills conditions 1–6 of Definition 6.

Theorem 3 now allows us to conclude that the translation preserves the full TCTL (or its safety fragment if $\mathcal{R}$ is only a one-by-many correspondence).

There are several reductions from TAPN to networks of timed automata that fit into the general framework [21, 32, 53] and the theory is applicable also to reductions between other time-dependent models including Time Petri nets [17, 23, 24, 34]. For more discussion we refer the reader to [32].

7.2 Translation from TAPN to Networks of Timed Automata

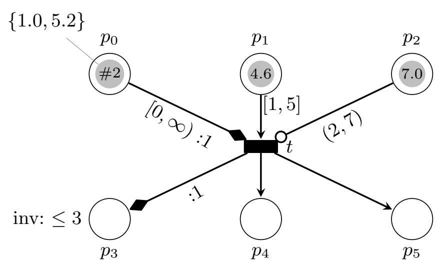
We will now present a translation from k-bounded TAPN (where the maximum number of tokens in every reachable marking is at most k) to networks of timed automata [7, 8] in the UPPAAL style (see e.g. [10] for an introduction to the formalism) in order to demonstrate the applicability of the framework described in the previous subsection.

For each token in the net, we create a parallel component in the network of timed automata. As the net is k-bounded, we will need at most k such components. In each of these parallel automata there is a location corresponding to each place in the net. Whenever a TA is in one of these locations, it simulates a token in the corresponding place. Moreover, each automaton has a local clock x which represents the age of the token. All automata simulating the tokens have the same structure, the only difference being their initial locations that correspond to the initial placement of tokens in the net. Because there may not always be exactly k tokens present during the execution of the net, we add a new location P_capacity to represent currently unused tokens.

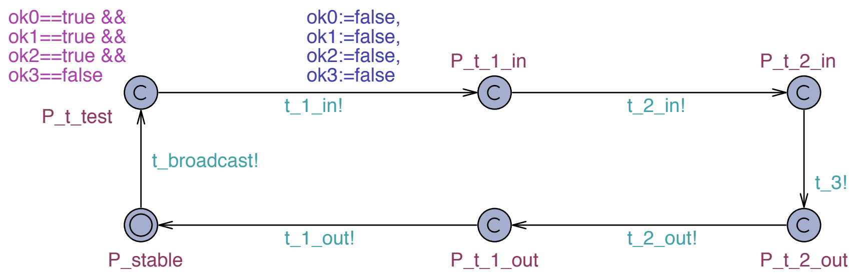
In addition to these 'token' automata we create a single control automaton. The purpose is to simulate the firing of transitions and to move tokens around via handshake synchronization initiated by the control automaton. This automaton has a location P_stable which acts as a mutex in the sense that the control automaton moves out of this location once the simulation of a transition begins and returns back once the simulation of the transition ends. Hence the proposition *stable* is defined as ($\mathsf{P\_stable} = 1$). Moreover, each time the automaton is in P_stable, the token automata in the composed UPPAAL network correspond to a marking in the TAPN. This directly implies that the generated TTS is *delay-preserves-stable* as time delay steps do not change the placement of tokens. We shall now demonstrate how the translation works on an example; the full algorithm is described in [31].

Consider the 5-bounded TAPN in Figure 9a. The translated network of UPPAAL timed automata is given below it. It consists of the control automaton in Figure 9b and five token automata like the one in Figure 9c. The token automata differ only in their initial locations, otherwise they are identical. Hence, in our example, we have two token automata whose initial locations are P0, and the remaining three have initial locations P1, P2 and P_capacity, respectively.

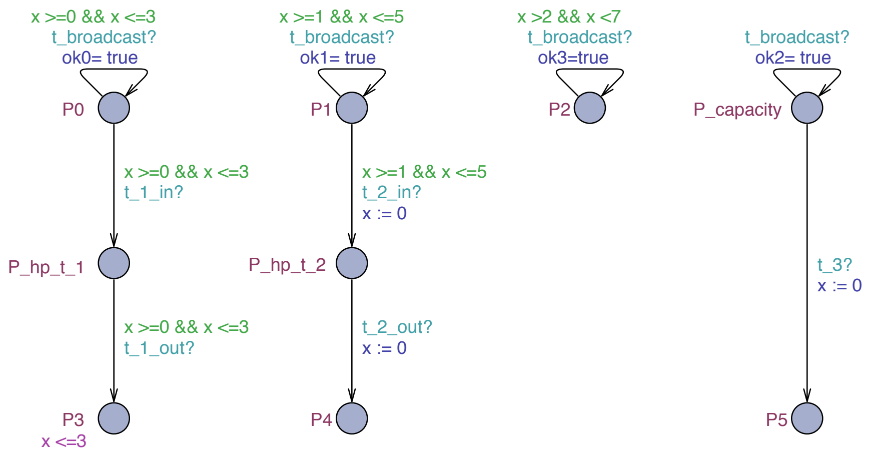
The communication in the network of timed automata begins when the controller broadcasts on the channel t_broadcast. All token automata that can accept the broadcast (i.e. their guards evaluate to true) will participate and set the



(a) A simple TAPN net



(b) Timed automaton controlling the firing of the transition t



(c) Timed automata templates for each token in the net with local clock x

Fig. 9: Translation from TAPN to UPPAAL network of timed automata

corresponding global boolean variables ok0, . . . , ok3 to true. The UPPAAL implementation of broadcast allows the controller to move to the location P_t_test only if the associated invariant where ok0, . . . , ok2 are all true and ok3 is false is satisfied, otherwise the broadcast cannot be executed. It is now clear that performing the broadcast is possible only if there is at least one token of an appropriate age in all input places of t, including P_capacity as a new token will be produced, and at the same time there is no token of age in the interval $(2, 7)$ in the place p_2. This is very important for the preservation of liveness TCTL properties, as once the transition firing is initiated, it should be always possibile to successfully finish it. Otherwise the generated transition system would not be *eventually-stable*. For this reason, the reader can notice that while the interval on the arc from p_0 to t is $[0, \infty)$, the corresponding guard in the token automaton on the edge from P0 to P_hp_t_1 requires the age of the token to be also less or equal to 3. The reason for this is that the token will be transported to the place p_3 and its age will be preserved. Any age value larger than 3 would violate the invariant in place p_3; hence as before the *eventually-stable* property might fail.

After the broadcast transition was successfully executed, then the effect of firing the transition t is simulated by a series of handshake synchronizations on channels t_1_in, t_2_in, t_3, t_2_out, t_1_out initiated by the controller and we have a guarantee that such a sequence will always bring the controller to the stable location P_stable, hence ensuring that the generated TTS is *eventually-stable*. Moreover, all locations of the controller are committed (do not allow any time delay steps), which means that the generated TTS is also *delay-implies-stable*.

The reason why the tokens are not moved directly to their destinations but are temporarily stored at the locations P_hp_t_1 and and P_hp_t_2 is to avoid the situation where a newly produced token is immediately consumed by firing of the same transition (as it may happen if the transition shared some input and output places).

In case the net contains more transitions, the controller automaton contains a similar loop for all such transitions. This concludes our example. It is relatively easy to argue (see [31] for details) that the original and the translated systems are in one-by-many equivalence. This gives us a polynomial time reduction from the full TCTL model checking problem of timed-arc Petri nets to TCTL model checking problem on networks of timed automata.

We have implemented the reduction described in this section in the open source verification tool TAPAAL [1]. The tool provides a graphical user interface for modelling, simulation and verification of timed-arc Petri nets. Further, we have modelled our example of alternating bit protocol described in Section 4 in TAPAAL and verified the correctness of its behaviour by asking about the violation of synchronization property described in Example 1. As the protocol model is unbounded and contains invariants, automatic verification is not possible. Instead, we considered an under-approximation of the protocol behaviour by limiting the maximum number of messages in transit so that the net becomes bounded. The protocol does not violate the correctness property for any number of messages in transit that we were able to verify. The verification times are measured on an Intel®CPU @ 2.67GHz based computer with 4 GB of memory. The results for a different maximum number of messages in transit are compared in Table 2 with a manually created UPPAAL model of the protocol.

It is clear that both the UPPAAL and TAPAAL models experience the state-space explosion problem so that even relatively small instances take a long ver-

Messages	UPPAAL	TAPAAL
1	< 1s	< 1s
2	< 1s	< 1s
3	< 1s	1.4s
4	< 1s	16.3s
5	2.2s	165.3s
6	14.4s	-
7	141.9s	-

(a) ABP without symmetry reduction

Messages	UPPAAL	TAPAAL
9	2.5s	1.1s
10	3.6s	1.9s
11	10.9s	2.9s
12	24.7s	4.2s
13	89.0s	6.1s
14	239.3s	8.8s
15	-	12.8s

(b) ABP **with** symmetry reduction

Table 2: Verification of ABP; dashes indicate more than 5 minutes running time

ification time. Here the native UPPAAL model is verified faster than the one automatically translated to UPPAAL automata from the TAPN model. On the other hand, the models contain lots of symmetric behaviour, so we also verified both models with symmetry reduction activated. Here, on the other hand, TAPAAL translation provides significantly faster verification compared to the native UPPAAL model. A similar story is true also for a few other experiments we ran and it seems to be connected to the fact that even though the translated models are larger than the manually created UPPAAL models, TAPAAL better exploits the benefits of symmetry reduction. A more detailed investigation of this phenomenon is a part of the future research.

8 Conclusion

In this article we provided an overview of decidability and complexity results related to verification of timed-arc Petri nets extended with transport arcs, age invariants and inhibitor arcs. We described a general framework for arguing when a translation between two time-dependent models preserves TCTL model checking and provided an example of such a translation from timed-arc Petri nets to networks of timed automata. The initial experimental data look promising and in the future we shall consider larger case studies and invest a significant effort into further development of the tool TAPAAL, including its own verification engine.

Among the different extensions of Petri nets with time aspects, we believe that timed-arc Petri nets constitute a convenient modeling formalism and with the recent development of its tool support, TAPN will become an attractive alternative to other modeling approaches.

References

[1] TAPAAL. http://www.tapaal.net.
[2] UPPAAL. http://www.uppaal.com.

[3] P.A. Abdulla and A. Nylén. Better is better than well: On efficient verification of infinite-state systems. *Proceedings of 15th Annual IEEE Symposium on Logic in Computer Science (LICS'00)*, pages 132–140, 2000.

[4] P.A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *Proceedings of the 22nd International Conference on Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *LNCS*, pages 53–70. Springer-Verlag, 2001.

[5] P.A. Abdulla, J. Deneux, P. Mahata, and A. Nylén. Forward reachability analysis of timed Petri nets. In *Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, volume 3253 of *LNCS*, pages 343–362. Springer-Verlag, 2004.

[6] P.A. Abdulla, P. Mahata, and R. Mayr. Dense-timed Petri nets: Checking zenoness, token liveness and boundedness. *Logical Methods in Computer Science*, 3(1):1–61, 2007.

[7] R. Alur and D. Dill. Automata for modelling real-time systems. In *Proceedings of the 17th International Colloquium on Algorithms, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer-Verlag, 1990.

[8] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[9] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.

[10] G. Behrmann, A. David, and K.G. Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04)*, number 3185 in LNCS, pages 200–236. Springer-Verlag, 2004.

[11] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. Software Eng.*, 17(3):259–273, 1991.

[12] B. Berthomieu, P-O. Ribet, and F. Vernadat. The tool TINA — construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, 2004.

[13] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between timed automata and bounded time Petri nets. In *Proc. of FORMATS'06*, volume 4202 of *LNCS*, pages 82–97. Springer, 2006.

[14] T. Bolognesi and P. Cremonese. The weakness of some timed models for concurrent systems. Technical Report CNUCE C89-29, CNUCE–C.N.R., 1989.

[15] T. Bolognesi, F. Lucidi, and S. Trigila. From timed Petri nets to timed LOTOS. In *Proceedings of the IFIP WG 6.1 Tenth International Symposium on Protocol Specification, Testing and Verification (Ottawa 1990)*, pages 1–14. North-Holland, Amsterdam, 1990.

[16] H. Boucheneb, G. Gardey, and O.H. Roux. TCTL model checking of time Petri nets. *Journal of Logic and Computation*, 19(6):1509–1540, 2009.

[17] P. Bouyer, S. Haddad, and P.-A. Reynier. Timed Petri nets and timed automata: On the discriminating power of zeno sequences. *Information and Computation*, 206(1):73–107, 2008.

[18] F.D.J. Bowden. Modelling time in Petri nets. In *Proceedings of the Second Australia-Japan Workshop on Stochastic Models*, 1996.

[19] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proceedings of the 10th International Conference on Computer-Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 546–550. Springer-Verlag, 1998.

[20] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF toolset. In *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM-RT'04)*, volume 3185 of *LNCS*, pages 237–267. Springer-Verlag, 2004.

[21] J. Byg, K.Y. Jørgensen, and J. Srba. An efficient translation of timed-arc Petri nets to networks of timed automata. In *Proc. of ICFEM'09*, volume 5885 of *LNCS*, pages 698–716. Springer, 2009.

[22] J. Byg, K.Y. Jørgensen, and J. Srba. TAPAAL: Editor, simulator and verifier of timed-arc Petri nets. In *Proc. of ATVA'09*, volume 5799 of *LNCS*, pages 84–89. Springer, 2009.

[23] F. Cassez and O.H. Roux. Structural translation from time Petri nets to timed automata. *ENTCS*, 128(6):145 – 160, 2005. Proc. of AVoCS'04.

[24] J.S. Dong, P. Hao, S. Qin, J. Sun, and W. Yi. Timed Automata Patterns. *IEEE Transactions on Software Engingeering*, 34(6):844–859, 2008.

[25] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[26] G. Gardey, D. Lime, M. Magnin, and O.H. Roux. Romeo: A tool for analyzing time Petri nets. In *Proc. of CAV'05*, volume 3576 of *LNCS*, pages 418–423. Springer, 2005.

[27] M. Hack. Petri Net Language. Technical Report MIT-LCS-TR-159, Massachusetts Institute of Technology, Cambridge, MA, USA, 1976.

[28] H.M. Hanisch. Analysis of place/transition nets with timed-arcs and its application to batch process control. In *Proceedings of the 14th International Conference on Application and Theory of Petri Nets (ICATPN'93)*, volume 691 of *LNCS*, pages 282–299, 1993.

[29] F. Heitmann, D. Moldt, K.H. Mortensen, and H. Rölke. Petri nets tools database quick overview. `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html`, Accessed: 28.10.2010.

[30] L. Jacobsen, M. Jacobsen, and M. H. Møller. Undecidability of coverability and boundedness for timed-arc Petri nets with invariants. In *Proc. of MEMICS'09*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009. ISBN 978-3-939897-15-6.

[31] L. Jacobsen, M. Jacobsen, and M.H. Møller. Modelling and verification of timed-arc Petri nets. Master's thesis, Department of Computer Science, Aalborg University, Denmark, 2010. Available at `http://tapaal.net`.

[32] L. Jacobsen, M. Jacobsen, M.H. Møller, and J. Srba. A framework for relating timed transition systems and preserving TCTL model checking. In *Proceedings of the 7th European Performance Engineering Workshop (EPEW'10)*, volume 6342 of *LNCS*, pages 83–98. Springer-Verlag, 2010.

[33] L. Jacobsen, M. Jacobsen, M.H. Møller, and J. Srba. A framework for relating timed transition systems and preserving TCTL model checking. Technical Report FIMU-RS-2010-09, Faculty of Informatics, Masaryk Univ., 2010.

[34] A. Janowska, P. Janowski, and D. Wróblewski. Translation of Intermediate Language to Timed Automata with Discrete Data. *Fundamenta Informaticae*, 85(1-4):235–248, 2008.

[35] L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.

[36] F. Laroussinie and K.G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII)*, pages 439–456. Kluwer, B.V., 1998.

[37] E.W. Mayr. An algorithm for the general Petri net reachability problem (preliminary version). In *Proceedings of the 13th Ann. ACM Symposium on Theory of Computing*, pages 238–246. Assoc. for Computing Machinery, 1981.

[38] Ph. M. Merlin. *A Study of the Recoverability of Computing Systems.* PhD thesis, University of California, Irvine, CA, USA, 1974.

[39] P.M. Merlin and D.J. Faber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.

[40] M.L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

[41] M. Nielsen, V. Sassone, and J. Srba. Properties of distributed timed-arc Petri nets. In *Proceedings of the 21st International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01)*, volume 2245 of *LNCS*, pages 280–291. Springer-Verlag, 2001.

[42] F.L. Pelayo, F. Cuartero, V. Valero, H. Macia, and M.L. Pelayo. Applying timed-arc Petri nets to improve the performance of the MPEG-2 encoding algorithm. In *Proceedings of the 10th International Multimedia Modelling Conference (MMM'04)*, pages 49–56. IEEE Computer Society, 2004.

[43] F.L. Pelayo, F. Cuartero, V. Valero, M.L. Pelayo, and M.G. Merayo. How does the memory work? by timed-arc Petri nets. In *Proceedings of the 4th IEEE International Conference on Cognitive Informatics (ICCI'05)*, pages 128–135, 2005.

[44] W. Penczek and A. Pólrola. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach.* Springer-Verlag, 2006.

[45] C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Darmstadt, 1962.

[46] C. Ramchandani. *Performance Evaluation of Asynchronous Concurrent Systems by Timed Petri Nets.* PhD thesis, Massachusetts Institute of Technology, Cambridge, 1973.

[47] V.V. Ruiz, F. Cuartero Gomez, and D. de Frutos Escrig. On non-decidability of reachability for timed-arc Petri nets. In *Proceedings of the 8th International Workshop on Petri Net and Performance Models (PNPM'99)*, pages 188–196, 1999.

[48] V.V. Ruiz, D. de Frutos Escrig, and O. Marroquin Alonso. Decidability of properties of timed-arc Petri nets. In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets (ICATPN'00)*, volume 1825 of *LNCS*, pages 187–206. Springer-Verlag, 2000.

[49] V.V. Ruiz, J.J. Pardo, and F. Cuartero. Translating TPAL specifications into timed-arc Petri nets. In *Proceedings of the 23rd International Conference on Applications and Theory of Petri Nets (ICATPN'02)*, volume 2360 of *LNCS*, pages 414–433. Springer-Verlag, 2002.

[50] V.V. Ruiz, F.L. Pelayo, F. Cuartero, and D. Cazorla. Specification and analysis of the MPEG-2 video encoder with timed-arc Petri nets. *Electronic Notes Theoretial Computer Science*, 66(2), 2002.

[51] J. Sifakis. Use of Petri nets for performance evaluation. In *Proceedings of the Third International Symposium IFIP W.G. 7.3., Measuring, Modelling and Evaluating Computer Systems (Bonn-Bad Godesberg, 1977)*, pages 75–93. Elsevier Science Publishers, Amsterdam, 1977.

[52] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proceedings of the 13th Annual Symposim on Theoretical Aspects of Computer Science (STACS'96)*, volume 1046 of *LNCS*, pages 347–359. Springer-Verlag, 1996.

[53] J. Srba. Timed-arc Petri nets vs. networks of timed automata. In *Proceedings of the 26th International Conference on Application and Theory of Petri Nets (ICATPN 2005)*, volume 3536 of *LNCS*, pages 385–402. Springer-Verlag, 2005.

[54] J. Srba. Comparing the expressiveness of timed automata and timed extensions of Petri nets. In *Proc. of FORMATS'08*, volume 5215 of *LNCS*, pages 15–32. Springer, 2008.

[55] J. Wang. *Timed Petri Nets, Theory and Application.* Kluwer Academic Publishers, 1998. ISBN ISBN 0-7923-8270-6.