Interval Abstraction Refinement for Model Checking of Timed-Arc Petri Nets

Sine Viesmose Birch, Thomas Stig Jacobsen, Jacob Jon Jensen,
Christoffer Moesgaard, Niels Nørgaard Samuelsen, and Jiří Srba

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

Abstract. State-space explosion is a major obstacle in verification of time-critical distributed systems. An important factor with a negative influence on the tractability of the analysis is the size of constants that clocks are compared to. This problem is particularly accented in explicit state-space exploration techniques. We suggest an approximation method for reducing the size of constants present in the model. The proposed method is developed for Timed-Arc Petri Nets and creates an under-approximation or an over-approximation of the model behaviour. The verification of approximated Petri net models can be considerably faster but it does not in general guarantee conclusive answers. We implement the algorithms within the open-source model checker TAPAAL and demonstrate on a number of experiments that our approximation techniques often result in a significant speed-up of the verification.

1 Introduction

Formal verification of time-dependent systems has been an active area of research for the last two decades or so. There are two prominent models that involve timing: timed automata (TA) [1] and different time extensions of Petri nets like Time Petri Nets (TPN) [22] and Timed-Arc Petri Nets (TAPN) [5,15]. Both symbolic[1] and explicit time-representation techniques have been developed for these models. For TA and TPN, it is well known [4,25] that the explicit (discrete-time) semantics coincide up to reachability with the continuous (real-time) semantics on time models with closed (non-strict) clock guards. A similar result can be proved also for TAPNs. The state-space exploration techniques for continuous semantics usually rely on symbolic zone-based abstractions (using the DBM data structure [12]). On the other hand, the discrete state-spaces can be searched in a direct manner where the clock values are remembered explicitly. The explicit approach can successfully compete with the symbolic one, as long as the constants in clock guards are reasonably small [6,19,17,3,16]. As the sizes of constants grow, the models become increasingly more difficult to verify, in particularly in case of explicit verification techniques.

[1] Referring here to a symbolic way to represent clock values and not to symbolic techniques based on decision diagrams.

As a motivating example, consider a design of a task scheduling algorithm for an embedded system where timing constraints are obtained from real physical measurements given in nanoseconds. Here a worst-case and best-case execution time of a certain task can be in the interval from 117 to 185 nanoseconds, while having the period of 10000 nanoseconds (the timing is taken from the model of a LEGO Mindstorm scheduling algorithm [14] created by software engineering students at Aalborg University). If a model of the task scheduling algorithm is populated with a larger number of components at this precision level, checking for the schedulability becomes quickly intractable. However, we may instead of the measured values approximate that the task duration is between 1 to 2 time units with the period of 100 units, abstracting away the precise timing and hence extending the task execution window. In case we succeed to verify that the system is schedulable under this abstraction (as it is the case for the LEGO scheduler), the schedulability of the original system is established as well.

Our contribution is a methodology that allows us to perform automatically such abstractions. The technique is demonstrated on the model of timed-arc Petri nets. The main idea is that time intervals of the form $[a, b]$, where $a \leq b$ are nonnegative integers, can be divided by a given approximation constant r and become $[\lfloor a/r \rfloor, \lceil b/r \rceil]$ in case of over-approximation and $[\lceil a/r \rceil, \lfloor b/r \rfloor]$ in case of under-approximation. By doing this, the constants used in the net are reduced, resulting possibly in large (even exponential) savings in verification time and memory. However, over-approximated nets allow for more behaviour while the under-approximated ones contain less behaviour and this may result in inconclusive verification answers. We discuss the correctness of the approximation techniques in the continuous as well as discrete semantics and both for the reachability and liveness properties. The approximation algorithms are implemented in a publicly available, open-source model checker TAPAAL [10], including a suitable GUI support, and we demonstrate its applicability on a number of case studies, ranging from academic examples to real-world inspired scenarios. For example in the LEGO case study [14], it takes 3366 seconds (more than 56 minutes) to verify that all tasks meet their deadlines, while if we over-approximate the intervals by dividing them with $r = 10$ it takes 36 seconds and with $r = 50$ only 7 seconds, still providing conclusive answers.

Related Work. Abstraction techniques like over-approximation [9] and under-approximation [21,24] have been studied in the past, including a counter-example guided abstraction methodology [8] where spurious counter-examples are used to refine the current approximation. Our approach is inspired by these techniques but focuses exclusively on the refinement of timing information and efficient feasibility analysis of the generated traces. State equations [23] and linear programming are often used to over-approximate the reachable space-space of untimed Petri nets. This technique is efficient, however, the timing information is completely disregarded, resulting often in inconclusive answers for timed nets. The authors in [13] suggest an algorithm for under- and over-approximations of timed safety automata by approximating the union operation on zones. Our method is not based on zones and it is targeted instead towards explicit state-space

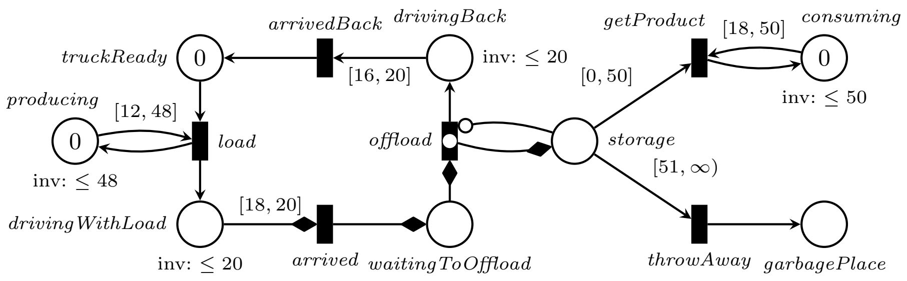


Fig. 1. Producer/consumer running example (intervals $[0, \infty]$ are not drawn)

exploration techniques where it can be combined with some recently introduced techniques and data structures like PTrie and Time Darts [16]. Finally, a time-relaxing method for a network of automata where events have interval-durations is described in [2]. The work proposes a pseudo-polynomial algorithm that enlarges delay intervals so that constants can be divided by a large greatest-common divisor (gcd). However, the division by gcd is, perhaps surprisingly, not a sound operation for liveness properties in the discrete semantics as we show in Section 3.2. Also, the method in [2] assumes that the network of automata satisfy the language intersection property (the language of the network is equal to the intersection of languages of the individual components). Our model of timed-arc Petri nets is more general as it supports also urgency, age invariants and inhibitor arcs (the language intersection property is not preserved anymore) and our approximation algorithm is simpler (with polynomial running time) and at the same time the experiments document a promising performance. Last but not least, another contribution of our work is the integration of the approximation algorithms into the tool TAPAAL.

2 Definitions

We start by informally introducing timed-arc Petri nets using a running example in Figure 1. The net consists of eight places (circles) and six transitions (rectangles) and models a simple producer/consumer system where produced items are loaded on a truck, transported to an off-load storage and later processed by the consumer, while at the same time the truck returns to the producer site. The producer, consumer and the truck are represented by three tokens in the depicted marking, all having the initial age 0. In the initial marking the transition *load* is not enabled because its input places do not contain tokens of ages that fit into the time intervals on the input arcs of the transition (by agreement we will not draw intervals of the form $[0, \infty]$ that do not restrict the ages of tokens in any way). However, if we wait for between 12 to 48 hours (longer delay than 48 hours is not allowed due to the age invariant ≤ 48 associated with the place *producing*), the transition *load* can fire. The firing consumes the two tokens from the input places and produces two fresh tokens of age 0 into the places *producing* and *drivingWithLoad*. Now after another 18 to 20 hours the track arrives (by firing the transition *arrived*) to its destination. As the pair of arcs (with diamond-shaped tips) connected with the transition *arrived* are the so-called transport arcs, the token from *dirivingWithLoad* is transported into the place *waitingToOffload* and its age is preserved. Similarly once the transition *offload* is fired, the age of the token moved into the place *storage* now represents the total time the product was in transfer. Note that the transition *offload* has a special dot in the middle, meaning that it is urgent and once it is enabled, time cannot progress any more (though the transition does not have priority over other enabled transitions in the net). Moreover, *offload* cannot fire as long as there is a token in the place *storage* due to the inhibitor arc with a circle-shaped tip. Finally, the truck starts its journey back to the producer and the product can be consumed by the consumer. In case that the total amount of time the product was in transport exceeds 50 hours, it cannot be consumed any more and can only be thrown away while marking the place *garbagePlace*. The model can also contain weighted arcs (not depicted in our figure) that will consume/produce multiple tokens along the same arc.

We now proceed with a formal definition of timed-arc Petri nets (TAPN). Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, $\mathbb{N}_0^\infty = \mathbb{N}_0 \cup \{\infty\}$ and $\mathbb{R}^{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$. We define the set of well-formed 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 a subset of $\mathcal{I}$ used in invariants as $\mathcal{I}^{\text{inv}} = \{[0, b] \mid b \in \mathbb{N}_0^\infty\}$.

Definition 1. *A TAPN is a tuple $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I)$ where*

- *P is a finite set of places,*
- *T is a finite set of transition such that $P \cap T = \emptyset$,*
- *T_{Urgent} is a finite set of urgent transitions such that $T_{Urgent} \subseteq T$,*
- *$IA \subseteq P \times T$ is a finite set of input arcs,*
- *$OA \subseteq T \times P$ is a finite set of output arcs,*
- *$g : IA \to \mathcal{I}$ is a time constraint function assigning guards to input arcs,*
- *$w : IA \cup OA \to \mathbb{N}$ is a function assigning weights to input and output arcs,*
- *$Type : IA \cup OA \to Types$ is a type function assigning a type to all arcs, where $Types = \{Normal, Inhib\} \cup \{Transport_j \mid j \in \mathbb{N}\}$ such that*
  - *if $Type(a) = Inhib$ then $a \in IA$,*
  - *if $Type((p, t)) = Transport_j$ for some $(p, t) \in IA$ then there is exactly one $(t, p') \in OA$ such that $Type((t, p')) = Transport_j$ and $w((p, t)) = w((t, p'))$,*
  - *if $Type((t, p')) = Transport_j$ for some $(t, p') \in OA$ then there is exactly one $(p, t) \in IA$ such that $Type((p, t)) = Transport_j$ and $w((p, t)) = w((t, p'))$,*
- *$I : P \to \mathcal{I}^{inv}$ is a function assigning age invariants to places.*

The preset of input places of a transition $t \in T$ is defined as ${}^\bullet t = \{p \in P \mid (p, t) \in IA, Type((p, t)) \neq Inhib\}$. Similarly, the postset of output places of t is defined as $t^\bullet = \{p \in P \mid (t, p) \in OA\}$. Let $\mathcal{B}(\mathbb{R}^{\geq 0})$ be the set of all finite multisets over $\mathbb{R}^{\geq 0}$. A *marking* M on N is a function $M : P \to \mathcal{B}(\mathbb{R}^{\geq 0})$ where for every place $p \in P$ and every token $x \in M(p)$ we have $x \in I(p)$.

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

We say that a transition $t \in T$ is enabled in a marking M by the multisets of tokens $\mathit{In} = \{(p, x_p^1), (p, x_p^2), \ldots, (p, x_p^{w((p,t))}) \mid p \in {}^\bullet t\} \subseteq M$ and $\mathit{Out} = \{(p', x_{p'}^1), (p', x_{p'}^2), \ldots, (p', x_{p'}^{w((p',t))}) \mid p' \in t^\bullet\}$ if

1. for all input arcs except inhibitor arcs, the tokens from *In* satisfy the age guards of the arcs, i.e.
   $\forall (p,t) \in \mathit{IA}.\mathit{Type}((p,t)) \neq \mathit{Inhib} \Rightarrow x_p^i \in g((p,t))$ for $1 \leq i \leq w((p,t))$
2. for any inhibitor arc pointing from a place p to the transition t, the number of tokens in p satisfying the guard is smaller than the weight of the arc, i.e.
   $\forall (p,t) \in \mathit{IA}.\mathit{Type}((p,t)) = \mathit{Inhib} \Rightarrow |\{x \in M(p) \mid x \in g((p,t))\}| < w((p,t))$
3. for all input and output arcs that 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.
   $\forall (p,t) \in \mathit{IA}.\forall (t,p') \in \mathit{OA}.\mathit{Type}((p,t)) = \mathit{Type}((t,p')) = \mathit{Transport}_j \Rightarrow (x_p^i = x_{p'}^i \wedge x_{p'}^i \in I(p'))$ for $1 \leq i \leq w((p,t))$
4. for all output arcs that are not part of a transport arc, the age of the output token is 0, i.e.
   $\forall (t,p') \in \mathit{OA}.\mathit{Type}((t,p') = \mathit{Normal} \Rightarrow x_{p'}^i = 0$ for $1 \leq i \leq w((p,t))$.

A TAPN N defines a timed transition system where states are 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 $M' = (M \setminus \mathit{In}) \uplus \mathit{Out}$ 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 there does not exist any $t \in T_{Urgent}$ and any d', $0 \leq d' < d$, such that t becomes enabled after the time delay d' (by delaying d time units no token violates any of the age invariants and the delay can at most last until the first urgent transition becomes enabled). 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.

We have just defined a *continuous semantics* of TAPNs where the possible time delays are from the domain of nonnegative real numbers. By restricting the delays only to nonnegative integers, we get the *discrete semantics* of TAPNs.

We write $M \to M'$ if either $M \xrightarrow{d} M'$ or $M \xrightarrow{t} M'$ for some delay d or a transition firing t. We write $M \xrightarrow{d,t} M'$ if there is a marking M'' such that $M \xrightarrow{d} M''$ and $M'' \xrightarrow{t} M'$. A *maximum* run of a net N from the initial marking M_0 is any infinite alternating sequence $M_0 \xrightarrow{d_0,t_0} M_1 \xrightarrow{d_1,t_1} M_2 \xrightarrow{d_2,t_2} \cdots$ or a finite alternating sequence $M_0 \xrightarrow{d_0,t_0} M_1 \xrightarrow{d_1,t_1} M_2 \xrightarrow{d_2,t_2} \cdots \xrightarrow{d_{n-1},t_{n-1}} M_n$ where either (i) for any delay $d \geq 0$ there is a marking M_d such that $M_n \xrightarrow{d} M_d$ or (ii) there is a delay $d \geq 0$ such that $M_n \xrightarrow{d} M_d$ and M_d does not allow any further nonzero delay and M_d does not enable any transition.

Let φ be a boolean combination of atomic predicates of the form $p \bowtie n$ where $p \in P$, $\bowtie \in \{=, <, >, \leq, \geq\}$ and $n \in \mathbb{N}_0$ (such predicates compare the number of tokens in a place p against the constant n), and the predicate *deadlock*. The satisfability of a formula φ in a marking M is defined by $M \models p \bowtie n$ if $|M(p)| \bowtie n$, and $M \models$ *deadlock* if there is no delay d and no transition t such that $M \xrightarrow{d,t} M'$. The extension to boolean operators is obvious. A formula φ is *deadlock-free* if it does not contain any proposition *deadlock*.

We can now define the reachability (EF) and liveness (EG) questions, as supported by the tool TAPAAL, for a given marked TAPN (N, M_0).

Definition 2 (Reachability). *We write* $(N, M_0) \models EF\,\varphi$ *if there is a computation* $M_0 \rightarrow^* M$ *such that* $M \models \varphi$.

Definition 3 (Liveness). *We write* $(N, M_0) \models EG\,\varphi$ *if there is a maximum run such that all markings* M *on this run satisfy* $M \models \varphi$.

If M_0 is clear from the context, we write only $N \models EF\,\varphi$ or $N \models EG\,\varphi$. The dual operators $AG\,\varphi \equiv \neg EF \neg\varphi$ and $AF\,\varphi \equiv \neg EG \neg\varphi$ are defined as expected.

Remark 1. If a query $EF\varphi$ is satisfied, the evidence for this fact is a finite run ending in a marking satisfying φ. The witness for the formula $EG\varphi$ is a maximum run invariantly satisfying φ. The maximum run is either finite (ending in a marking where we can delay forever or in a marking where no transition firing and no delay is possible) or infinite. If such an infinite run exists then there is also one that has a lasso shape (see e.g [3]) so that the sequence of the transition firings is of the form $t_1 t_2 \ldots (t_\ell \ldots t_n)^\omega$.

3 Interval Abstractions

Verification of reachability and liveness queries is a computationally hard problem because the size of the reachable state-space can be exponential compared to the size of the analyzed net. For timed systems, there are two sources of this exponential explosion. The first one is that Petri nets allow to model parallel activities that can have exponentially many different interleavings. The second degree of explosion stems from the addition of timing aspects. In this section, we shall see how the explosion caused by the timing constraints can be greatly reduced while still providing conclusive answers in many concrete scenarios. We suggest two approximation methods, one creating an over-approximation and the other one an under-approximation. Both methods rely on a given approximation constant r that determines the ratio by which the constants in the net are scaled. As constants in a net must be integers, we need to round the scaled values. For over-approximation, we enlarge the available intervals in the net, while for under-approximation we shrink them. A special care has to be given to inhibitor arcs as they inhibit behaviour. Hence for over-approximation we need to shrink the intervals on inhibitor arcs while for under-approximation we do the opposite.

Definition 4 (Interval abstraction by over-approximation). *Let* $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I)$ *be a TAPN, let* M_0 *be its initial marking and let* r *be a positive natural number (approximation constant). The over-approximation algorithm on an input* (N, M_0) *outputs a marked net* (N_r^{over}, M_0) *where* $N_r^{over} = (P, T, T_{Urgent}, IA', OA, g', w', Type', I')$ *such that*

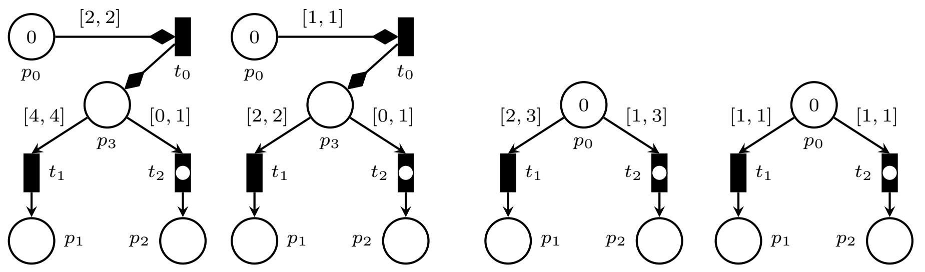
- $IA' = IA \setminus \{(p,t) \in IA \mid Type((p,t)) = Inhib, \lceil \frac{a}{r} \rceil > \lfloor \frac{b}{r} \rfloor \text{ where } [a,b] = g((p,t))\}$
- $g'((p,t)) = \begin{cases} [\lfloor \frac{a}{r} \rfloor, \lceil \frac{b}{r} \rceil] \text{ if } g((p,t)) = [a,b] \text{ and } Type((p,t)) \neq Inhib \\ [\lceil \frac{a}{r} \rceil, \lfloor \frac{b}{r} \rfloor] \text{ if } g((p,t)) = [a,b] \text{ and } Type((p,t)) = Inhib \end{cases}$ *for all* $(p,t) \in IA'$,
- $w'(x,y) = w(x,y)$ *and* $Type'(x,y) = Type(x,y)$ *for all* $(x,y) \in IA' \cup OA$,
- $I'(p) = [0, \lceil \frac{b}{r} \rceil]$ *where* $[0,b] = I(p)$ *for all* $p \in P$.

The over-approximation clearly runs in polynomial time. Note that the over-approximation may remove some inhibitor arcs in case that the resulting interval is empty (meaning that the lower-bound is larger than the upper-bound).

Definition 5 (Interval abstraction by under-approximation). *Let* $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I)$ *be a TAPN, let* M_0 *be its initial marking and let* r *be a positive natural number (approximation constant). The under-approximation algorithm on an input* (N, M_0) *outputs a marked net* (N_r^{under}, M_0) *where* $N_r^{under} = (P, T', T'_{Urgent}, IA', OA', g', w', Type', I')$. *Let* $X = \{(p,t) \in IA \mid Type((p,t)) \neq Inhib, \lceil \frac{a}{r} \rceil > \lfloor \frac{b}{r} \rfloor \text{ where } [a,b] = g((p,t))\}$. *Then*

- $T' = T \setminus \{t \in T \mid (p,t) \in X \text{ for some } p \in P\}$,
- $T'_{Urgent} = T_{Urgent} \setminus \{t \in T \mid (p,t) \in X \text{ for some } p \in P\}$,
- $IA' = IA \setminus X$
- $OA' = OA \setminus \{(t,p) \in OA \mid t \in T \setminus T'\}$,
- $g'((p,t)) = \begin{cases} [\lceil \frac{a}{r} \rceil, \lfloor \frac{b}{r} \rfloor] \text{ if } g((p,t)) = [a,b] \text{ and } Type((p,t)) \neq Inhib \\ [\lfloor \frac{a}{r} \rfloor, \lceil \frac{b}{r} \rceil] \text{ if } g((p,t)) = [a,b] \text{ and } Type((p,t)) = Inhib \end{cases}$ *for all* $(p,t) \in IA'$,
- $w'(x,y) = w(x,y)$ *and* $Type'(x,y) = Type(x,y)$ *for all* $(x,y) \in IA' \cup OA'$,
- $I'(p) = [0, \lfloor \frac{b}{r} \rfloor]$ *where* $[0,b] = I(p)$ *for all* $p \in P$.

The under-approximation clearly runs in polynomial time. Observe that the construction of under-approximated net slightly differs from the over-approximated net. In particular, if an arc of a transition is removed because of an empty interval, then it is necessary to remove also the connected transition as otherwise the net might achieve more behaviour.



(a) $N \models EF\, p_1{=}1$ and $N_2^{over} \not\models EF\, p_1{=}1$ **(b)** $N \not\models EF\, p_1{=}1$ and $N_3^{under} \models EF\, p_1{=}1$

Fig. 2. Urgent transitions with timed input arcs

3.1 Approximation Correctness for Reachability

In order to argue about the correctness of the over-approximation for reachability queries, we wish to prove that if $N \models EF\, \varphi$ then also $N_r^{over} \models EF\, \varphi$ for any $r \geq 1$. For under-approximation, the implication should be the other way round. The correctness clearly does not hold if the formula φ contains any deadlock proposition as the approximations can both create new deadlocks and remove some existing ones. Moreover, the situation is a slightly more complicated than it may look, as urgent transitions with time-guarded input arcs may also influence the answer to reachability queries as demonstrated in Figure 2. This is caused by the fact that once the interval on an urgent transitions is approximated, it may disable a time delay that was possible in the original net. Hence for example the net N in Figure 2a can mark the place p_1 while this is not possible in the over-approximated net N_2^{over} because once the token of age 1 arrives to the place p_3, no time delay is allowed due to the urgency of t_2. This means that t_1 is never enabled in the over-approximated net. A similar situation can be observed also for the under-approximated net in Figure 2b.

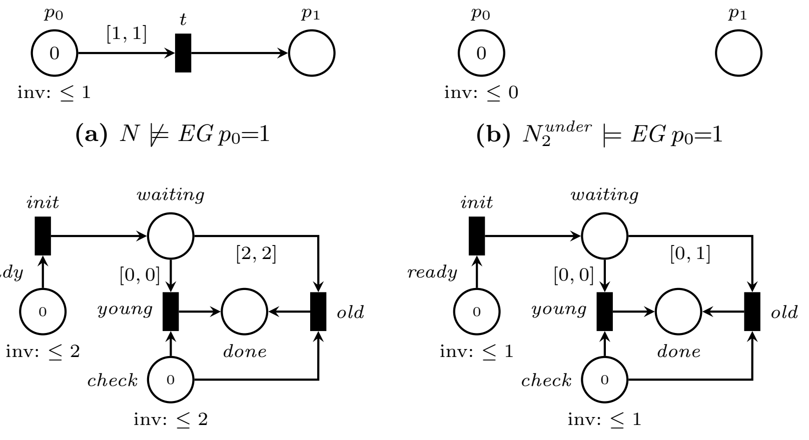
We can now prove that the approximations are correct for any deadlock-free reachability objective, assuming that urgent transitions have only trivial guards on incoming arcs[2]. The following correctness theorem holds both for the continuous as well as the discrete semantics.

Theorem 1. *Let $N = (P, T, T_{Urgent}, IA, OA, g, w, Type, I)$ such that $g((p,t)) = [0, \infty]$ for all $t \in T_{Urgent}$ and let φ be a deadlock-free formula. If $N \models EF\, \varphi$ then $N_r^{over} \models EF\, \varphi$ for any $r \geq 1$. If $N_r^{under} \models EF\, \varphi$ for some $r \geq 1$ then $N \models EF\, \varphi$.*

3.2 Approximation Correctness for Liveness

Let us first notice that we cannot expect to prove under-approximation correctness for liveness queries as under-approximation can introduce additional deadlocks that can create non-existent maximum runs. For example, consider the net

[2] This restriction on urgent transitions also guarantees that DBM-based algorithms can be used in the TAPAAL continuous engine [11].



(a) $N \not\models EG\, p_0{=}1$ **(b)** $N_2^{under} \models EG\, p_0{=}1$

(c) $N \models EG\, done{=}0$ **(d)** $N_3^{over} \not\models EG\, done{=}0$

Fig. 3. Problem with over-approximation and under-approximation for liveness

N in Figure 3a that does not satisfy the query $EG\, p_0{=}1$ as any maximum run is forced to fire the transition t. On the other hand, the under-approximated net for $r = 2$ in Figure 3b clearly satisfies the query.

A less expected message is that the same problem is present also for the over-approximation as relaxing the net behaviour can remove some existing deadlocks. Consider the TAPN in Figure 3c that satisfies $EG\, done{=}0$ by the maximum run *delay 1*, *init*, *delay 1* that actually only uses integer delays. The over-approximated net for $r = 3$ in Figure 3d cannot deadlock in a similar situation as before as any maximum run will necessarily place a token into the place *done* (both in discrete and continuous semantics). Hence $N_3^{over} \not\models EG\, done{=}0$. For the discrete semantics we get already for $r = 2$ (greatest common divisor of all constants in the net) that $N_2^{over} \not\models EG\, done{=}0$.

To sum up, even though the over- and under-approximations are correct for reachability objectives, the correctness does not hold any more for liveness queries. Nevertheless, in the next section we show that we can still efficiently verify whether the maximal runs for EG queries in the approximated models are valid maximal runs also in the original ones.

4 Trace Validation

The aim of this section is to define the so-called trace net. A trace net guides the state-space search in the original net based on a given sequence of transitions (trace). The use of a trace net is to efficiently verify whether a trace proposed by a net approximation is executable in the original net or not (for each trace we construct a different trace net). Assume now a fixed untimed trace of the form $trace = t_1 t_2 \ldots t_n$ or $trace = t_1 t_2 \ldots (t_\ell \ldots t_n)^\omega$ where $t_i \in T$ for all i, $1 \leq i \leq n$. By $\#(t)$ we denote the number of occurrences of the transition t in $trace$ (for an infinite trace only it its finite prefix $t_1 \ldots t_n$). By $\#_i(t)$, where $1 \leq i \leq n$, we denote the number of occurrences of t in the prefix $t_1 \ldots t_i$ of $trace$.

For the given TAPN N, we shall now construct a TAPN N^{trace} that restricts the behaviour of the net N so that transitions can be executed only in the order that follows the sequence *trace* (without imposing any concrete time delays), while at the same time making sure that along any computation in N^{trace} the proposition deadlock evaluates equivalently as it would in the original net N.

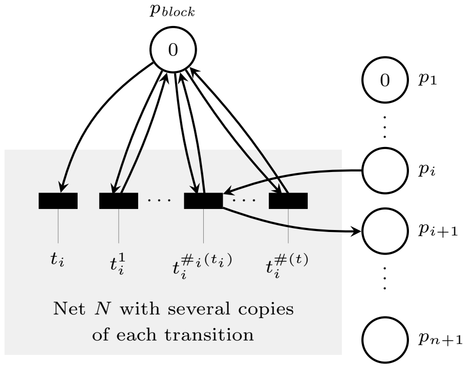
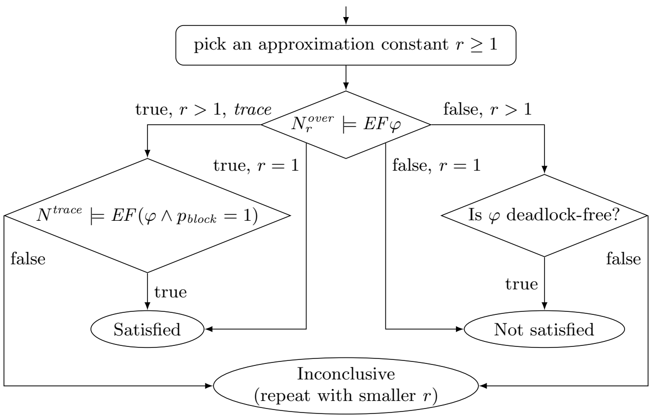


Fig. 4. Construction of the net N^{trace}

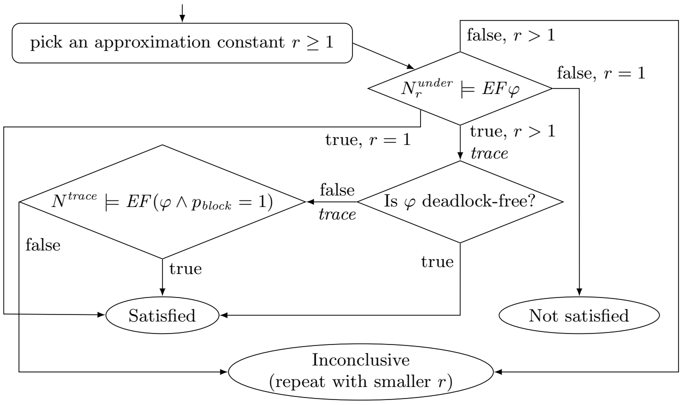
We shall modify the net N and its initial marking M_0 via the following steps until we get N^{trace} and the initial marking M_0^{trace}. The construction is depicted in Figure 4. In what follows, by a *simple arc* we mean a normal input or output arc of weight one; simple input arcs have the guard $[0, \infty]$.

- For each transition $t \in T$ we create $\#(t)$ additional copies of t, denoted by $t^1, t^2, \ldots, t^{\#(t)}$, such that every new copy t^j, $1 \leq j \leq \#(t)$, has an identical preset and postset as t (the same input and output places connected with arcs of the same type and with the same weight and containing the same time intervals as guards). The new copies of t are urgent if and only if t is urgent. Clearly adding these transitions does not have any effect on the behaviour of the net.
- We add new places $p_1, p_2, \ldots, p_n$ and a place p_{n+1} such that if *trace* is finite the p_{n+1} is a newly added place and if *trace* is infinite (of the form $t_1 t_2 \ldots (t_\ell \ldots t_n)^\omega$) then $p_{n+1} = p_\ell$. The added places have the age invariant $[0, \infty]$. There will be always exactly one token in the places $p_1, \ldots, p_{n+1}$, such that if the place p_i is marked then the only transition that can fire is some copy of t_i. In the marking M_0^{trace} the place p_1 contains one token and the places $p_2, \ldots, p_{n+1}$ are empty.
- For each i, $1 \leq i \leq n$, we add two simple arcs $(p_i, t_i^{\#_i(t_i)})$ and $(t_i^{\#_i(t_i)}, p_{i+1})$. In other words, the places p_i and p_{i+1} are connected via the next available copy of the transition t_i so that each copy is used only once in the sequence (note that the same transition can appear several times in *trace*). This construction imposes an order in which transitions can be fired, following step by step the sequence of transitions in *trace*. On the other hand, the modified net has a full freedom in choosing time delays as in the original net.
- Finally we add a place p_{block}, initially marked with a token, together with a pair of simple arcs (p_{block}, t_i^j) and (t_i^j, p_{block}) for each copy t_i^j of every transition t_i. We also add a simple arc (p_{block}, t_i) for every original transition t_i in *trace*. The purpose of p_{block} is to allow to deviate for one step from the transition sequence in *trace* so that every transition enabled in the original net N is enabled also in N^{trace}. However, once this step is taken (via firing some of the original transitions t_i), the token from p_{block} is consumed (and



pick an approximation constant $r \geq 1$
$N_r^{over} \models EF\varphi$
true, $r > 1$, *trace*
false, $r > 1$
true, $r = 1$
false, $r = 1$
$N^{trace} \models EF(\varphi \wedge p_{block} = 1)$
Is φ deadlock-free?
false
true
true
false
Satisfied
Not satisfied
Inconclusive (repeat with smaller r)

(a) Over-approximation flow diagram for $EF\varphi$



pick an approximation constant $r \geq 1$
false, $r > 1$
false, $r = 1$
$N_r^{under} \models EF\varphi$
true, $r = 1$
true, $r > 1$ *trace*
$N^{trace} \models EF(\varphi \wedge p_{block} = 1)$
false *trace*
Is φ deadlock-free?
false
true
true
Satisfied
Not satisfied
Inconclusive (repeat with smaller r)

(b) Under-approximation flow diagram for $EF\varphi$

Fig. 5. Flow diagrams for over- and under-approximation reachability queries

the whole net N^{trace} terminates). This is to make sure that all enabled transitions in N are enabled also in N^{trace} in order to preserve the validity of the proposition *deadlock*.

Theorem 2. *Let N be a TAPN and let φ be a formula (possibly containing the proposition deadlock). If $N^{trace}, M_0^{trace} \models EF\,(\varphi \wedge p_{block} = 1)$ then $N, M_0 \models EF\,\varphi$. If $N^{trace}, M_0^{trace} \models EG\,(\varphi \wedge p_{block} = 1)$ then $N, M_0 \models EG\,\varphi$.*

Finally, we present the refinement process for approximation of reachability queries in Figure 5. The diagrams for liveness only differ in the point that a trace has to be always verified even for the under-approximation and in case a trace is not discovered in the approximated net, the answer is always inconclusive. The correctness of the flow diagrams follows from Theorem 1 and 2.

5 Evaluation

We discuss the case studies of *Patient Monitoring System (PMS)* [7] where the patient's pulse rate and oxygen saturation level is monitored and abnormal situations should be detected within given deadlines (constants scaled up to 250 seconds), *Business Activity with Participant Completion (BAwPC)* [18]—a web-service protocol from WS-BA where we verify its safety (avoidance of invalid states) using the fact that the original protocol is flawed while its enhanced variant is safe [18], *Train Level Crossing (TLC)*—a standard benchmark case study where trains are crossing a road and traffic lights should be controlled correctly, *Producer and Consumer Synchronization (PCS)*—our running example scaled by introducing more producers and consumers and *Plate Spinning Problem (PSP)* [20] where jugglers try to keep a number of plates spinning indefinitely. The greatest common divisor in all models is 1.

The approximations were implemented in the model checker TAPAAL available at `http://www.tapaal.net/`. The experiments, run on a Macbook Pro 2.7GHz Intel Core i7, were terminated once the memory usage exceeded 4GB (OOM) or the verification took longer than 5 minutes (🕒). In the summary table we report on the running time using TAPAAL's discrete verification engine [16] and the column labelled with $r = 1$ corresponds to verification where no approximation is used. The rows marked with "no trace" correspond to EF or EG queries that are not satisfied (and hence no trace is returned). Only over-approximation is used here as under-approximation cannot reach conclusive results in this case. The rows marked with "trace" are satisfied EF and EG queries returning a trace that is verified by the trace net[3]. An inconclusive answer is prefixed by a question mark. We also note for each row whether we used depth-first search (DFS) or breadth-first search (BFS) when exploring the approximated nets.

In case of singleton intervals on arcs, under-approximation will remove such arcs (including the connected transitions). This may quickly result in a net where too many transitions are missing and the verification answers become inconclusive. Our experiments show that if we instead keep the arcs with singleton intervals (divided by the approximation constant r and rounded down), then we are likely to get more conclusive answers. Of course, we are not creating an under-approximation any more. However, this is not an issue as for liveness queries the trace returned by under-approximation must be always verified by the trace net (see Section 3.2) and we can do the same also for the reachability queries. Our experimental data use the variant of under-approximation described above.

The experiments show that both approximations frequently provide conclusive answers and significantly speedup the verification process. The general trend is that increasing the approximation constant r improves the verification times up to a certain point after which the improvements are not that significant and finally may result in inconclusive answers (like in PSP) or a timeout (in case of

[3] For the PMS case-study only under-approximation is reported as over-approximation was returning inconclusive answers; the size scaling in PMS is also different for the satisfied and unsatisfied query in order to provide measurable data.

Patient Monitoring System (PMS) — Reachability

	Size	r=1	r=2	r=3	r=5	r=7	r=10
no trace over-approx. BFS	1	🕒	57.1 s	18.8 s	11.7 s	4.1 s	0.7 s
	2	🕒	102.3 s	25.4 s	38.6 s	5.6 s	0.8 s
	3	🕒	231.9 s	40.7 s	65.4 s	8.2 s	1.1 s
	4	🕒	🕒	67.4 s	135.0 s	56.0 s	1.5 s
trace under-approx. BFS	1	39.3 s	5.0 s	**?** 2.1 s	1.2 s	**?** 0.5 s	0.5 s
	2	242.5 s	21.9 s	**?** 4.5 s	2.2 s	**?** 0.7 s	1.2 s
	3	🕒	39.7 s	**?** 3.9 s	3.2 s	**?** 0.9 s	1.4 s
	4	🕒	50.4 s	**?** 4.7 s	4.2 s	**?** 1.3 s	0.8 s

Business Activity Protocol (BAwPC) — Reachability

	Size	r=1	r=2	r=4	r=6	r=10	r=15
no trace over-approx. DFS	1	🕒	88.6 s	16.4 s	7.8 s	3.1 s	2.3 s
	2	🕒	🕒	93.0 s	38.3 s	13.6 s	9.5 s
	3	🕒	🕒	🕒	136.5 s	41.1 s	31.7 s
	4	🕒	🕒	🕒	🕒	110.5 s	88.0 s
trace over-approx. DFS	1	1.3 s	0.4 s	0.2 s	0.1 s	0.2 s	0.2 s
	2	155.9 s	24.5 s	4.9 s	2.6 s	0.2 s	0.2 s
	3	🕒	🕒	114.6 s	42.5 s	0.3 s	0.3 s
	4	🕒	🕒	🕒	🕒	0.4 s	0.3 s
trace under-approx. DFS	1		0.4 s	0.2 s	0.1 s	0.1 s	0.1 s
	2		17.5 s	2.8 s	1.3 s	0.2 s	0.1 s
	3		🕒	51.1 s	14.6 s	0.2 s	0.2 s
	4		🕒	🕒	116.6 s	0.3 s	0.3 s

Train Level Crossing (TLC) — Reachability

	Size	r=1	r=2	r=3	r=5	r=7	r=9
no trace over-approx. BFS	1	5.7 s	0.8 s	0.3 s	0.1 s	0.1 s	0.0 s
	2	🕒	21.4 s	5.0 s	1.0 s	0.3 s	0.2 s
	3	🕒	🕒	96.2 s	10.0 s	2.4 s	1.1 s
	4	🕒	🕒	🕒	80.7 s	14.4 s	5.4 s
trace over-approx. BFS	1	4.1 s	1.9 s	1.6 s	1.4 s	1.4 s	1.4 s
	2	9.9 s	2.8 s	2.0 s	1.6 s	1.6 s	1.6 s
	3	12.0 s	3.0 s	2.1 s	1.6 s	1.6 s	1.6 s
	4	11.9 s	3.0 s	2.1 s	1.7 s	1.6 s	1.6 s
trace under-approx. BFS	1		0.9 s	0.6 s	**?** 0.1 s	1.3 s	0.5 s
	2		1.9 s	1.0 s	**?** 0.7 s	1.5 s	0.5 s
	3		2.0 s	0.9 s	**?** 6.2 s	1.5 s	0.5 s
	4		2.0 s	0.9 s	**?** 50.4 s	1.5 s	0.5 s

Producer and Consumer Synchronization (PCS) — Liveness

	Size	r=1	r=2	r=3	r=4	r=5	r=6
trace over-approx. DFS	1	0.8 s	1.0 s	1.0 s	🕒	**?** 171.1 s	🕒
	2	10.9 s	6.8 s	6.4 s	🕒	🕒	🕒
	3	126.0 s	30.4 s	27.5 s	🕒	🕒	🕒
	4	🕒	162.2 s	142.9 s	🕒	🕒	🕒
trace under-approx. DFS	1		1.0 s	0.3 s	0.2 s	0.7 s	0.3 s
	2		6.7 s	1.2 s	1.1 s	4.2 s	1.0 s
	3		30.2 s	7.6 s	7.3 s	22.9 s	6.7 s
	4		157.2 s	47.1 s	50.3 s	120.3 s	43.4 s

Plate Spinning Problem (PSP) — Liveness

	Size	r=1	r=2	r=3	r=4	r=5	r=6
trace over-approx. DFS	1	12.4 s	0.6 s	0.3 s	0.3 s	0.3 s	**?** 0.1 s
	2	41.3 s	1.5 s	0.4 s	0.4 s	0.4 s	**?** 0.1 s
	3	100.3 s	3.2 s	0.7 s	0.7 s	0.7 s	**?** 0.1 s
	4	213.2 s	6.3 s	1.1 s	1.1 s	1.1 s	**?** 0.1 s
trace under-approx. DFS	1		1.7 s	0.5 s	0.6 s	🕒	0.3 s
	2		5.5 s	1.5 s	1.5 s	0.4 s	0.4 s
	3		12.8 s	3.2 s	3.2 s	2.2 s	0.7 s
	4		26.8 s	6.3 s	6.2 s	1.2 s	1.1 s

an over-approximation that suddenly allows too much behaviour like in PCS). Occasionally, inconclusive answers may appear for relatively small r values (like for PMS where $r = 3$ and $r = 7$) due to an unfortunate rounding of guards that produces infeasible traces.

As already mentioned, the reported experiments rely on the discrete-time engine. We also investigated how the approximation methods behaved in case of a continuous TAPAAL engine that performs a zone-based exploration (using DBM data structure). The general observation in most of such experiments is that the approximations do not significantly influence the verification times that usually differ by a constant factor only. This is caused by the fact that the continuous engine performs a symbolic exploration that is not that affected by the size of the constants like during the explicit exploration. The comparison of discrete vs. continuous verification is not in the scope of this paper and we refer to [6,19,16] for further discussion.

6 Conclusion

We provided a simple, yet efficient method for discrete-time verification of timed-arc Petri net. The approximation algorithms were implemented in the tool TAPAAL and the experiments document a high practical applicability, in particular for nets where the timing constraints are robust, meaning that small changes in the guard intervals do not change the validity of the properties in question. As a result, the designers of formal models do not have to consider so carefully the size of constants in their models anymore; in many cases the constants can be automatically lowered while still providing conclusive answers.

References

1. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Alur, R., Itai, A., Kurshan, R., Yannakakis, M.: Timing verification by successive approximation. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 137–150. Springer, Heidelberg (1993)
3. Andersen, M., Gatten Larsen, H., Srba, J., Grund Sørensen, M., 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. Asarin, E., Maler, O., Pnueli, A.: On discretization of delays in timed automata and digital circuits. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 470–484. Springer, Heidelberg (1998)
5. Bolognesi, T., Lucidi, F., Trigila, S.: From timed Petri nets to timed LOTOS. In: IFIP WG 6.1 Tenth International Symposium on Protocol Specification, Testing and Verification, pp. 1–14. North-Holland, Amsterdam (1990)
6. Bozga, M., Maler, O., Tripakis, S.: Efficient verification of timed automata using dense and discrete time semantics. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 125–141. Springer, Heidelberg (1999)

7. Cicirelli, F., Furfaro, A., Nigro, L.: Model checking time-dependent system specifications using time stream Petri nets and UPPAAL. Applied Mathematics and Computation 218(16), 8160–8186 (2012)
8. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
9. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. 16(5), 1512–1542 (1994)
10. 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)
11. David, A., Jacobsen, L., Jacobsen, M., Srba, J.: A forward reachability algorithm for bounded timed-arc Petri nets. In: SSV 2012. EPTCS, vol. 102, pp. 125–140. Open Publishing Association (2012)
12. 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)
13. Dill, D.L., Wong-Toi, H.: Verification of real-time systems by successive over and under approximation. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 409–422. Springer, Heidelberg (1995)
14. Drægert, A., Kaysen, A.C., Byrdal Kjær, J., Mikkelsen, F.B., Nduru, C., Petersen, D.S.: LEGO car safety systems. 5th Semester Software Engineer Project Report, Aalborg University (2014)
15. Hanisch, H.M.: Analysis of place/transition nets with timed-arcs and its application to batch process control. In: Ajmone Marsan, M. (ed.) ICATPN 1993. LNCS, vol. 691, pp. 282–299. Springer, Heidelberg (1993)
16. 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)
17. Jørgensen, K.Y., Larsen, K.G., Srba, J.: Time-darts: A data structure for verification of closed timed automata. In: SSV 2012. EPTCS, vol. 102, pp. 141–155. Open Publishing Association (2012)
18. Marques Jr., A.P., Ravn, A.P., Srba, J., Vighio, S.: Model-checking web services business activity protocols. International Journal on Software Tools for Technology Transfer (STTT) 15(2), 125–147 (2013)
19. Lamport, L.: Real-time model checking is really simple. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 162–175. Springer, Heidelberg (2005)
20. Larsen, K.G., Behrmann, G., Skou, A.: Exercises for UPPAAL (2008), `http://www.cs.aau.dk/~bnielsen/TOV08/ESV04/exercises`
21. Lee, W., Pardo, A., Jang, J.-Y., Hachtel, G., Somenzi, F.: Tearing based automatic abstraction for CTL model checking. In: ICCAD 1996, pp. 76–81. IEEE Computer Society (1996)
22. Merlin, P.M., Faber, D.J.: Recoverability of communication protocols: Implications of a theoretical study. IEEE Trans. on Comm. 24(9), 1036–1043 (1976)
23. Murata, T.: State equation, controllability, and maximal matchings of Petri nets. IEEE Trans. on Automatic Control 22(3), 412–416 (1977)
24. Pardo, A., Hachtel, G.D.: Incremental CTL model checking using BDD subsetting. In: DAC 1998, pp. 457–462. ACM (1998)
25. Popova-Zeugmann, L.: On time Petri nets. Elektronische Informationsverarbeitung und Kybernetik 27(4), 227–244 (1991)