Undecidability of Coverability and Boundedness for Timed-Arc Petri Nets with Invariants

Lasse Jacobsen, Morten Jacobsen and Mikael H. Møller

Department of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300,
9220 Aalborg Øst, Denmark
{lassejac,mortenja,mikaelhm}@cs.aau.dk

Abstract. Timed-Arc Petri Nets (TAPN) is a well studied extension of the classical Petri net model where tokens are decorated with real numbers that represent their age. Unlike reachability, which is known to be undecidable for TAPN, boundedness and coverability remain decidable. The model is supported by a recent tool called TAPAAL which, among others, further extends TAPN with invariants on places in order to model urgency. The decidability of boundedness and coverability for this extended model has not yet been considered. We present a reduction from two-counter Minsky machines to TAPN with invariants to show that both the boundedness and coverability problems are undecidable.

1 Introduction

Time-dependent models have been extensively studied due to increasing demands on the reliability and safety of embedded software systems. *Timed automata* [11] and various time-extensions of *Petri nets* (e.g. [4]) are among the most studied time-dependent models. A recent paper by Srba [14] provides a comparative overview of these models.

Timed-Arc Petri Nets (TAPN's) [4] is a popular time-extension of Petri Nets [10] in which each token is assigned an age (a real number), and time intervals on arcs restrict the ages of tokens that can be used to fire a transition. The reachability problem has been shown undecidable for TAPN [12]. In particular, a TAPN cannot correctly simulate a test for zero on a counter [3]. However, other problems, like boundedness and coverability remain decidable [2][1].

Recent work on the verification tool TAPAAL by Byg et al. [5] have, among other things, introduced invariants on places into the TAPN model as a way to represent urgency. However, urgency alone does not allow a TAPN to correctly simulate a test for zero on a counter. Nevertheless, we show that invariants on places makes the coverability and boundedness problem undecidable. We adopt the main idea from [12] (see also [6] for a similar proof technique for another time extension of Petri nets), in which a two-counter Minsky machine (2-CM) is weakly simulated by a TAPN. In contrast to their reduction, the extension of invariants allows us to detect when the net incorrectly simulates the 2-CM. Further, our reduction allows us to prove the undecidability of both the coverability and boundedness problems.

2 Basic Definitions

Many of the definitions in this section are following [13]. The set of all *time intervals* $\mathcal{I}$ and the set of time intervals for invariants $\mathcal{I}_{Inv}$ are defined according to the following abstract syntaxes where $a \in \mathbb{N}_0$, $b \in \mathbb{N}$ and $a < b$:

$$I ::= [a,a] \mid [a,b] \mid [a,b) \mid (a,b] \mid (a,b) \mid [a,\infty) \mid (a,\infty)$$
$$I_{Inv} ::= [0,0] \mid [0,b] \mid [0,b) \mid [0,\infty)$$

We define the predicate $r \in I$ for $r \in \mathbb{R}_0^+$ in the expected way.

Definition 1 (Timed-Arc Petri Net with Invariants). *A* Timed-Arc Petri Net with Invariants *(ITAPN) is a 5-tuple $N = (P, T, F, c, \iota)$ where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, $c : F|_{P \times T} \to \mathcal{I}$ is a function assigning time intervals to arcs from places to transitions, and $\iota : P \to \mathcal{I}_{Inv}$ is a function assigning invariants to places.*

We let $\mathcal{B}(\mathbb{R}_0^+)$ denote the set of finite multisets over $\mathbb{R}_0^+$. For a $B \in \mathcal{B}(\mathbb{R}_0^+)$ and some $d \in \mathbb{R}_0^+$, we let $B + d = \{b + d \mid b \in B\}$. Notationally, we use multisets as ordinary sets with the operations $\cup, \setminus, \subseteq, \in$ implicitly interpreted over multisets.

Let us now define a marking on a ITAPN.

Definition 2 (Marking). *A* marking *M on a ITAPN $N = (P, T, F, c, \iota)$ is a function $M : P \to \mathcal{B}(\mathbb{R}_0^+)$, such that for every place $p \in P$ it holds that for every token $x \in M(p)$, $x \in \iota(p)$. The set of all markings over N is denoted $\mathcal{M}(N)$.*

A *marked* ITAPN is a pair (N, M_0) where N is a ITAPN and M_0 is the initial marking. We only allow initial markings in which all tokens have age 0.

The *preset* of a transition t is ${}^\bullet t = \{p \in P \mid (p,t) \in F\}$ and the *postset* of t is $t^\bullet = \{p \in P \mid (t,p) \in F\}$.

Definition 3 (Firing rule). *Let $N = (P, T, F, c, \iota)$ be a ITAPN, M some marking on it and $t \in T$ be a transition of N.*

We say that t is enabled *if and only if $\forall p \in {}^\bullet t.\ \exists x \in M(p).\ x \in c(p,t)$, i.e. there is a token with an appropriate age at every place in the preset of t.*

If t is enabled in M, it can be fired*, whereby we reach a marking M' defined by $\forall p \in P.\ M'(p) = \big(M(p) \setminus C_t^-(p)\big) \cup C_t^+(p)$ (note that all operations are on multisets and there may be multiple choices for the sets $C_t^-(p)$ and $C_t^+(p)$ for each p. We simply fix the sets before firing t), where*

- $C_t^-(p) = \begin{cases} \{x\} & \text{if } p \in {}^\bullet t \wedge x \in M(p) \wedge x \in c(p,t) \\ \emptyset & \text{otherwise} \end{cases}$
- $C_t^+(p) = \begin{cases} \{0\} & \text{if } p \in t^\bullet \\ \emptyset & \text{otherwise} \end{cases}$

i.e. from each place $p \in {}^\bullet t$ we remove a token with an appropriate age, and we add a new token with age 0 to every $p \in t^\bullet$.

Definition 4 (Time delays). *Let $N = (P, T, F, c, \iota)$ be a ITAPN and M some marking on it. A time delay $d \in \mathbb{R}_0^+$ is allowed if and only if $(x + d) \in \iota(p)$ for all $p \in P$ and $x \in M(p)$, i.e. by delaying d time units no token violates the invariants. By delaying d time units we reach a marking M', defined as $M'(p) = M(p) + d$ for all $p \in P$.*

A marked ITAPN (N, M_0) is said to be *k-bounded* if the number of tokens in each place does not exceed k for any marking reachable from M_0. A marked ITAPN is *bounded* if it is k-bounded for some $k \in \mathbb{N}$.

Problem 1 (Boundedness). Given a marked ITAPN is it bounded?

A marking M on a ITAPN (N, M_0) is said to be *coverable* if there exists a marking M', reachable from M_0, s.t. $M'(p) \supseteq M(p)$ for each place p in the net.

Problem 2 (Coverability). Given a marked ITAPN (N, M_0) and some marking M, is M coverable?

3 Undecidability of Boundedness and Coverability

In this section we will prove the undecidability of boundedness and coverability by reduction from two-counter Minsky machines.

Definition 5. *A* Two-Counter Minsky Machine (2-CM) *with two non-negative registers r_1 and r_2 is a sequence of instructions $(I_1 : Ins_1;\ I_2 : Ins_2;\ \ldots\ I_{e-1} : Ins_{e-1};\ I_e : HALT)$ where for every j, $1 \leq j < e$, Ins_j is one of the two types:*

- *$r_i := r_i + 1;$ goto I_k; where $i \in \{1, 2\}$ and $k \in \{1, 2, \ldots, e\}$ (Increment).*
- *if $r_i > 0$ then $r_i := r_i - 1;$ goto $I_k;$ else goto $I_\ell;$ where $i \in \{1, 2\}$ and $k, \ell \in \{1, 2, \ldots, e\}$ (Test and decrement).*

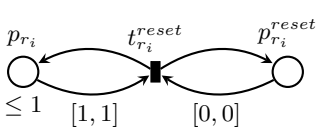
The last instruction is always the HALT instruction. A configuration *of a 2-CM is a triple (j, v_1, v_2) where $j \in \{1, 2, \ldots, e\}$ is the index of instruction I_j to be executed and v_1 and v_2 are the values of the registers r_1 and r_2, respectively.*

The computational step relation of a 2-CM is defined as expected and we use the notation $(j, v_1, v_2) \rightarrow (j', v_1', v_2')$ to denote that we perform the current instruction I_j with values v_1 and v_2 in the registers, resulting in the configuration (j', v_1', v_2').

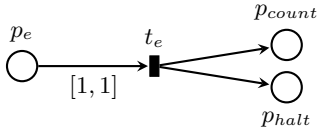
Definition 6 (The Halting Problem for 2-CM). *Given a 2-CM, is it possible to reach the halt instruction from the initial configuration $(1, 0, 0)$, i.e. $(1, 0, 0) \rightarrow^* (e, v_1, v_2)$ for some $v_1, v_2 \in \mathbb{N}_0$?*

Theorem 1 (Minsky [9]). *The halting problem for 2-CM is undecidable.*

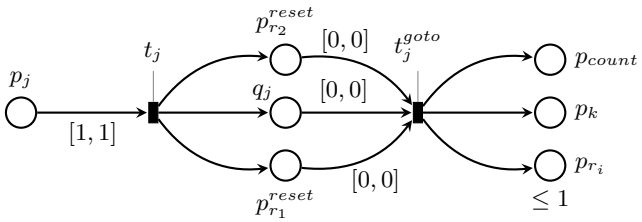
We will now describe the reduction from 2-CM to ITAPN. Given a 2-CM $(I_1 : Ins_1;\ I_2 : Ins_2;\ \ldots\ I_{e-1} : Ins_{e-1};\ I_e : HALT)$ we construct a ITAPN (P, T, F, c, ι) where



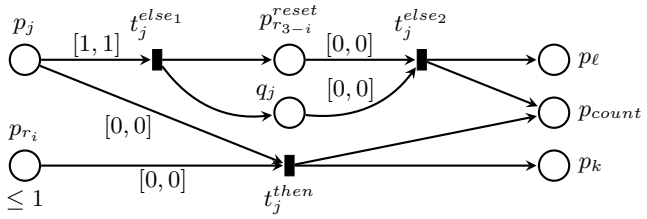
(a) Simulation of a register.



(b) Simulation of Halt instruction.



(c) Simulation of $I_j : r_i := r_i + 1;$ goto I_k.



(d) Simulation of I_j : if $r_i > 0$ then $r_i := r_i - 1;$ goto $I_k;$ else goto I_ℓ.

Fig. 1: ITAPN models for 2-CM simulation.

- $P = \{p_j, q_j \mid 1 \leq j < e\} \cup \{p_{r_1}, p_{r_1}^{reset}, p_{r_2}, p_{r_2}^{reset}\} \cup \{p_{count}\} \cup \{p_e, p_{halt}\}$
- $T = \{t_{r_1}^{reset}, t_{r_2}^{reset}\} \cup \{t_j, t_j^{goto} \mid Ins_j \text{ is of type increment}\} \cup \left\{t_j^{else_1}, t_j^{else_2}, t_j^{then} \mid Ins_j \text{ is of type test and decrement}\right\} \cup \{t_e\}$

The number of tokens in p_{r_1} and p_{r_2} correspond to the values of r_1 and r_2, the number of tokens in p_{count} remembers the number of computation steps which have been simulated in the net and $p_1, \ldots, p_e$ corresponds to the instructions $Ins_1, \ldots, Ins_e$ such that the place p_j contains one token if and only if the current instruction is Ins_j. For the flow relation we will split it into 4 parts.

- F_1 contains the arcs for the registers. For each register $r_i,\ i \in \{1, 2\}$, we add the following arcs to F_1

  $(p_{r_i}, t_{r_i}^{reset}),\ (t_{r_i}^{reset}, p_{r_i}),\ (p_{r_i}^{reset}, t_{r_i}^{reset}),\ (t_{r_i}^{reset}, p_{r_i}^{reset})$ where
  $c((p_{r_i}, t_{r_i}^{reset})) = [1, 1],\ \ c((p_{r_i}^{reset}, t_{r_i}^{reset})) = [0, 0]$ and $\iota(p_{r_i}) = [0, 1]$.

  This is illustrated in Figure 1a. The number of tokens on p_{r_i} indicates the value of the register. Notice the invariant on the register which disallows tokens with an age greater than 1. Placing a token on $p_{r_i}^{reset}$ allows us to reset the age of all tokens of age 1 in the register.

– F_2 contains the arcs for the increment instructions. For each increment instruction $I_j : r_i := r_i + 1;$ goto I_k;, we add the following arcs to F_2

(p_j, t_j), $(t_j, p_{r_2}^{reset})$, (t_j, q_j), $(t_j, p_{r_1}^{reset})$, $(p_{r_2}^{reset}, t_j^{goto})$, (q_j, t_j^{goto}), $(p_{r_1}^{reset}, t_j^{goto})$, (t_j^{goto}, p_{count}), (t_j^{goto}, p_k), (t_j^{goto}, p_{r_i}) where $c((p_j, t_j)) = [1, 1]$, $c((p_{r_2}^{reset}, t_j^{goto})) = [0, 0]$, $c((q_j, t_j^{goto})) = [0, 0]$ and $c((p_{r_1}^{reset}, t_j^{goto})) = [0, 0]$.

This is illustrated in Figure 1c. Notice that we require a delay of one time unit before firing t_j. Because of this, we allow tokens in each register to be reset (by placing tokens on $p_{r_1}^{reset}$ and $p_{r_2}^{reset}$). Following this, by firing t_j^{goto} a token is added to p_{count}, register r_i is incremented by adding a token to p_{r_i} and control is given to the next instruction I_k by placing a token on p_k.

– F_3 contains the arcs for the test and decrement instructions. For each test and decrement instruction $I_j :$ if $r_i > 0$ then $r_i := r_i - 1;$ goto $I_k;$ else goto I_ℓ;, we add the following arcs to F_3

$(p_j, t_j^{else_1})$, (p_j, t_j^{then}), (p_{r_i}, t_j^{then}), $(t_j^{else_1}, q_j)$, $(t_j^{else_1}, p_{r_{3-i}}^{reset})$, $(q_j, t_j^{else_2})$, $(p_{r_{3-i}}^{reset}, t_j^{else_2})$, $(t_j^{else_2}, p_\ell)$, $(t_j^{else_2}, p_{count})$, (t_j^{then}, p_{count}), (t_j^{then}, p_k) where $c((p_j, t_j^{else_1})) = [1, 1]$, $c((p_j, t_j^{then})) = [0, 0]$, $c((p_{r_i}, t_j^{then})) = [0, 0]$, $c((p_{r_{3-i}}^{reset}, t_j^{else_2})) = [0, 0]$ and $c((q_j, t_j^{else_2})) = [0, 0]$.

This is illustrated in Figure 1d. Notice that when we follow the else branch (firing transition $t_j^{else_1}$), we can only reset the ages of tokens in the register on which we are not testing for emptyness.

– F_4 contains the arcs for the HALT instruction. Formally it is defined as

$$F_4 = \{(p_e, t_e), (t_e, p_{count}), (t_e, p_{halt})\} \text{ where } c((p_e, t_e)) = [1, 1] .$$

This is illustrated in Figure 1b. Again we require a time delay of one time unit before t_e can be fired and a token placed at p_{halt}.

– The flow relation F can then be defined as the union of the four parts, i.e. $F = F_1 \cup F_2 \cup F_3 \cup F_4$ and we let $\iota(p) = [0, \infty)$ for all $p \in P \setminus \{p_{r_1}, p_{r_2}\}$.

We define the initial marking M_0 such that $M_0(p_1) = \{0\}$ and $M_0(p) = \emptyset$ for all $p \in P \setminus \{p_1\}$.

Let (N, M_0) be the marked ITAPN simulating a given 2-CM. Notice that every place in the net except for $p_{r_1}, p_{r_2}, p_{count}$ is *1-safe* (i.e. contains at most one token). In a *correct* simulation of the 2-CM by our net, a configuration (j, v_1, v_2) of the 2-CM corresponds to any marking M where

$$M(p_j) = \{0\}, \quad M(p_{r_i}) = \underbrace{\{0, 0, \ldots, 0\}}_{v_i \text{ times}} \text{ for } i \in \{1, 2\}, \tag{1}$$

$|M(p_{count})| = n$ where $n \in \mathbb{N}_0$ and $M(p) = \emptyset$ for all $p \in P \setminus \{p_j, p_{r_1}, p_{r_2}, p_{count}\}$.

We will now describe how to simulate the three types of instructions of a 2-CM in a *correct* way. Assume there is a token of age 0 in p_j.

If I_j is an increment instruction, we need to delay for one time unit in order to enable t_j (see Figure 1c). Because we delayed one time unit, all tokens in the registers are now of age 1. In a correct simulation, we fire repeatedly transitions $t_{r_1}^{reset}$ and $t_{r_2}^{reset}$ until all tokens in p_{r_1} and p_{r_2} are of age 0. Note that it is possible to *cheat* in the simulation, as it is possible to leave some tokens of age 1 in p_{r_1} or p_{r_2} when firing t_j^{goto}.

If I_j is a test and decrement instruction there are two possibilities (see Figure 1d). If there is a token of age 0 at p_{r_i}, we fire t_j^{then} in order to decrement the number of tokens in register r_i, and hand over the control to I_k by placing a token on p_k. Otherwise, in the correct simulation we delay one time unit before firing $t_j^{else_1}$. Then we reset the age of all the tokens in the other register, $p_{r_{3-i}}$ to 0. We then proceed by firing $t_j^{else_2}$. This will hand over control to instruction I_ℓ by placing a token on p_ℓ. Again note that it is possible to *cheat* in the simulation, either by leaving tokens of age 1 at $p_{r_{3-i}}$ when proceeding to the next instruction or by taking the *else*-branch even though there is a token at p_{r_i} (because the net does not force us to fire transition t_j^{then} when it is enabled).

If I_j is the halt instruction, we delay one time unit before we fire the last transition t_e and add a token to p_{halt}.

After every instruction one token is added to p_{count}. We will now prove a lemma detailing what happens if we cheat.

Lemma 1. *Let (j, v_1, v_2) be the current configuration of a 2-CM CM, (N, M_0) the associated ITAPN and M a marking corresponding to (j, v_1, v_2) (see Equation 1). If the net cheats then during the simulation of CM in the next computation step it is not possible to simulate an increment instruction, go to the halt state, nor to take the else-branch of a test and decrement instruction. Further, the net can do at most $v_1 + v_2$ decrements before getting stuck.*

Proof. We can perform an incorrect simulation in two ways:

- If all tokens in p_{r_1} and p_{r_2} are not reset to age 0 in an increment or test and decrement instruction before going to the next instruction.
- In a test and decrement instruction, the net can fire the transition $t_j^{else_1}$ even if there is a token of age 0 in p_{r_i}. This is possible by delaying 1 time unit to enable the transition. However, this will result in the tokens in p_{r_i} having age 1 and these can not be reset before going to the next instruction.

In both cases we end up in a marking M' where there is at least one token of non-zero age in either p_{r_1} or p_{r_2}. Observe that the simulation of increment, halt and the else-branch of a test and decrement instruction all require a delay of 1 time unit (see Figure 1) which would violate the invariants $\iota(p_{r_1})$ or $\iota(p_{r_2})$. Thus, the only possibility is to take the then-branch of a test and decrement instruction. However, this is only possible as long as there are tokens of age 0 in p_{r_1} or p_{r_2}. There are v_1 and v_2 tokens in p_{r_1} and p_{r_2}, repectively. Thus, the net can do at most $v_1 + v_2$ decrements before getting stuck. □

3.1 Undecidability Results

First we prove the undecidability of the boundedness problem.

Lemma 2. *Given a 2-CM CM and the associated ITAPN (N, M_0), CM halts if and only if N is bounded.*

Proof. We start by proving that if N is bounded then CM halts. Assume that N is k-bounded. Further, assume by contradiction that CM does not halt. After simulating $k+1$ computational steps of CM correctly, the net will be in a marking M where $|M(p_{count})| = k+1$. This is a contradiction to the assumption that N is k-bounded.

Now we prove the implication in the other direction. Assume that CM halts in n steps. We will show that N is $2n$-bounded. If we simulate CM correctly, there will be at most n tokens at the registers, and exactly n tokens at p_{count}. Hence, the net must cheat in order to become unbounded. In the worst case, it cheats at the last step, when there are at most $n-1$ tokens in the registers and $n-1$ tokens at p_{count}. Then we have that the net is $2n$-bounded since there will be at most $2(n-1)$ tokens at p_{count} by Lemma 1. □

From Lemma 2 we conclude the following theorem.

Theorem 2. *The boundedness problem is undecidable for ITAPN.*

We now prove the undecidability of the coverability problem.

Lemma 3. *Let M be a marking such that $M(p_{halt}) = \{0\}$ and $M(p) = \emptyset$ for all $p \in P \setminus \{p_{halt}\}$. Given a 2-CM CM and the associated marked ITAPN (N, M_0), as defined above, CM halts if and only if M is coverable from M_0.*

Proof. First we prove that if CM halts then M is coverable from M_0. Assume that the CM halts. By simulating CM correctly in N, we can easily see that we reach a marking M', with a token in p_{halt}, hence $M'(p) \supseteq M(p)$ for all $p \in P$.

Now we prove that if M is coverable from M_0 then CM halts. Assume that M is coverable from M_0. By assumption there exists a reachable marking M' such that $M'(p) \supseteq M(p)$ for all $p \in P$. By definition of coverability, it holds that $0 \in M'(p_{halt})$ and by Lemma 1 this is only possible if we simulate CM correctly in the net, hence CM halts. □

From Lemma 3 we conclude the following theorem.

Theorem 3. *The coverability problem is undecidable for ITAPN.*

4 Conclusion

We proved that coverability and boundedness is undecidable for Time-Arcs Petri Nets with Invariants by reduction from two-counter Minsky machines. The following table shows a summary of known results about Petri Nets (PN). Our results are emphasized.

	Reachability	Boundedness	Coverability
PN	decidable [8]	decidable [7]	decidable [7]
TAPN	undecidable [12]	decidable [2]	decidable [1]
ITAPN	undecidable [12]	*undecidable*	*undecidable*

Acknowledgements

We would like to thank Jiří Srba, Mads Chr. Olesen, Kenneth Y. Jørgensen and the anonymous reviewers for their comments and suggestions.

References

[1] P. A. Abdulla and A. Nylén. Timed Petri Nets and BQOs. In *Proc. of ICATPN*, volume 2075 of *LNCS*, pages 53–70. Springer, 2001.

[2] 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.

[3] T. Bolognesi and P. Cremonese. The Weakness of some Timed Models for Concurrent Systems. Technical Report CNUCE C89-29, CNUCE-C.N.R., Oct. 1989.

[4] T. Bolognesi, F. Lucidi, and S. Trigila. From Timed Petri Nets to Timed LOTOS. In *Proc. of IFIP WG 6.1 PSVT X*, pages 395–408, Ottawa, Canada, 1990. North-Holland Publishing.

[5] 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 5799 of *LNCS*. Springer, Dec 2009. To appear (available at author's website).

[6] Neil D. Jones, Lawrence H. Landweber, and Y. Edmund Lien. Complexity of some problems in petri nets. *Theor. Comput. Sci.*, 4(3):277–299, 1977.

[7] R. M. Karp and R. E. Miller. Parallel Program Schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[8] E. W. Mayr. An Algorithm for the General Petri Net Reachability Problem. In *Proc. of STOC '81*, pages 238–246, Milwaukee, WI, USA, 1981. ACM.

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

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

[11] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[12] V. V. Ruiz, F. C. Gomez, and D. de Frutos-Escrig. On Non-Decidability of Reachability for Timed-Arc Petri Nets. In *Proc. of PNPM '99*, pages 188–196, Washington DC, USA, 1999. IEEE Computer Society.

[13] J. Srba. Timed-Arc Petri Nets vs. Networks of Timed Automata. In *Proc. of ICATPN'05*, volume 3536 of *LNCS*, pages 385–402. Springer, 2005.

[14] 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.