All True Concurrency Models Start with Petri Nets: A Personal Tribute to Carl Adam Petri

Wojciech Penczek

1 Starting from Petri Nets

Paraphrasing 'All roads lead to Rome' one might say 'All true concurrency models start with Petri nets.' So, in my case, Petri nets have been my first research topic. Initially, I investigated their algebraic properties, but later my work focused on verification of Petri nets using various model reduction techniques and symbolic model-checking approaches.

1.1 Petri Nets Have Always Been Around

My first contact with Petri nets was in 1984–1985 during my studies at Warsaw University of Technology and the University of Warsaw. Then I learned the basic definitions of this true concurrency model for distributed systems and could enjoy its attractive graphical representations.

In 1986 when I joined the Institute of Computer Science of the Polish Academy of Sciences, my work, supervised by Prof. Antoni Mazurkiewicz, was focused on prime event structures, traces, and trace systems. I have been impressed by the match between trace systems and condition-event systems, their elegant composition and decomposition methods, and how the trace theory is used as a tool for reasoning about the behavior of Petri nets [5]. Later, I could also see how extensions of traces,

W. Penczek (✉)
Institute of Computer Science, PAS, Warsaw, Poland

Siedlce University of Natural Sciences and Humanities, Institute of Computer Science (ICS), Siedlce, Poland
e-mail: penczek@ipipan.waw.pl

W. Reisig, G. Rozenberg (eds.), *Carl Adam Petri: Ideas, Personality, Impact*,
https://doi.org/10.1007/978-3-319-96154-5_24

such as infinite traces or semi-traces, are used for modeling behaviors of more involved variants of Petri nets.

1.2 Meeting Carl Adam Petri at GMD

My first foreign research trip was to the 'Gesellschaft fur Mathematik und Datenverarbeitung' (GMD, Society for Mathematics and Information Technology) in 1988. Invited by Prof. Ursula Goltz I spent a few weeks at GMD, having the privilege of meeting there famous people working on Petri nets. These were Wolfgang Reisig, Eike Best, Kurt Lautenbach, and finally I was introduced to Carl Adam Petri. This was a very short meeting and we only exchanged a few sentences, but I do remember this contact very well, being under a very strong impression of the personality of Prof. Petri who was kindly interested in my research. He asked about my scientific interests and the aim of my visit to GMD. This personal contact with Prof. Petri has strongly influenced my scientific career and the topics of my interest. Most of my further research results have been concerned with verification of Petri nets.

2 Verification of Time Petri Nets

My work on verification of (time) Petri nets started with several results concerning abstractions and partial order reductions [8, 9]. In 2001 these were the main methods for alleviating the state explosion problem of state spaces. Regardless of the true concurrency nature of Petri nets, their state spaces built on marking graphs may grow exponentially in the number of transitions or can be infinite in the case of time Petri nets. So, efficient verification was possible only after abstracting or reducing the state spaces. Many verification methods for time Petri nets were adapted from those developed for Timed Automata and vice versa [10]. One of the most important examples is minimization algorithms for time Petri nets [12] exploiting *partitioning refinement*. Then, we put forward methods based on bounded model checking using BDDs as well as SAT- and SMT-solvers, applied to both concrete and abstract state spaces [4, 6, 7, 11, 13].

In order to discuss verification methods of time Petri nets in more detail, it is useful to recall their definition.

Definition 1 A time Petri net (TPN) is a six-tuple $\mathcal{N} = (P, T, F, m^\circ, Eft, Lft)$, where

- $P = \{p_1, \ldots, p_{np}\}$ is a finite set of *places*,
- $T = \{t_1, \ldots, t_{nT}\}$ is a finite set of *transitions*, where $P \cap T = \emptyset$,
- $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the *flow function*, where $\mathbb{N}$ is the set of natural numbers, including 0,
- $m^0 : P \to \mathbb{N}$ is the *initial marking* of $\mathcal{N}$,

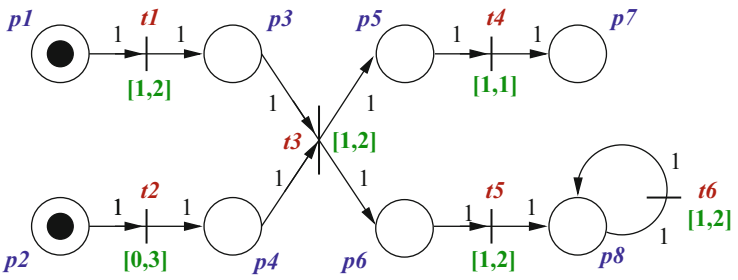

Fig. 1 A time Petri net

- $Eft: T \to \mathbb{N}, Lft: T \to \mathbb{N}U\{\infty\}$ are functions describing respectively the *earliest firing time* and the *latest firing time* of the transitions, where clearly $Eft(t) \leq Lft(t)$ for each $t \in T$.

A four-tuple (P, T, F, m^0) is called a Petri net.

An example of a time Petri net is shown in Fig. 1. The values of the functions Eft and Lft are given by $Eft(t) = 1$ and $Lft(t) = 2$ for $t \in \{t_1, t_3, t_5, t_6\}$, $Eft(t_2) = 0$, $Lft(t_2) = 3$, and $Eft(t_4) = Lft(t_4) = 1$.

Intuitively, a marking specifies how many tokens (denoted by the black dots) are stored in particular places. Firing of a transition consumes a number of tokens from its input places and produces a number of tokens in its output places; the above numbers are given by the corresponding values of F (represented in the figure by the numbers decorating the arrows connecting those pairs of a place and a transition for which the value of F is not equal to zero).

Intuitively, a transition t is enabled if all its input places contain a sufficient number of tokens to be consumed by the flow function. The earliest and latest firing times of a transition t specify the timing interval in which t can be fired. If the time passed since the transition t has become enabled reaches the value $Lft(t)$, the transition has to be fired, unless disabled by a firing of another transition. In order to simplify some subsequent notions, we consider only nets satisfying the following two conditions:

- the flow function F maps onto $\{0, 1\}$, so F can be identified with the relation $F \subseteq (P \times T) \cup (T \times P)$, and
- the number of tokens that can be stored in a place is limited to 1, so m^0 can be identified with the set of places m^0 assigns to 1. Introducing such a limit changes slightly the conditions under which a transition is enabled, which can be seen below.

In order to describe models for time Petri nets we need to give some basic definitions.

Let $t \in T$ be a transition.

– By the *preset* of t we mean the set of its input places $\bullet t = \{p \in P | F(p, t) = 1\}$.

- By the *postset* of t we mean the set of its output places $t\bullet = \{p \in P | F(t, p) = 1\}$.
- By a *marking* of $\mathcal{N}$ we mean any subset $m \subseteq P$.
- A transition t is *enabled* at m ($m[t\rangle$ for short) if $\bullet t \subseteq m$ and $t \bullet \cap (m \setminus \bullet t) = \emptyset$, i.e., m contains each input place of t, and m does not contain any output place of t except for these places which are both input and output places of t.
- By $en(m) = \{t \in T | m[t\rangle\}$ we mean the set of transitions enabled at m.
- If t is enabled at m, then t leads from m to the marking $m' = (m \setminus \bullet t) \cup t\bullet$.

2.1 Concrete Models of TPNs

A concrete state records a snapshot of the behavior of a TPN. In order to verify whether a system modeled by a TPN satisfies some properties, we need to represent the set of all its concrete states, defined below, together with their valuation, with propositions, which are used for formulating the properties. Such a structure is called a concrete model. Typically, one of two approaches to building concrete models is applied, depending on whether the flow of time is recorded by real variables, called clocks, or firing intervals. We discuss the clock approach in more detail.

A *concrete state* of a TPN is a pair $\sigma = (m, clock)$, where m is a marking and *clock* is a function that gives values to the clocks that can be associated with the transitions or the places. If a net is distributed, i.e., composed of well-defined sequential processes, then the clocks can be also assigned to its processes [10]. The explanation below is for the clocks assigned to the transitions. The initial state is denoted by $\sigma^0 = (m^0, (0, \ldots, 0))$. The concrete state changes because of either the firing of an enabled transition $t \in T$ for which $Eft(t) \leq clock(t) \leq Lft(t)$, denoted $\sigma \xrightarrow{t}_c \sigma'$, or the passing of some time provided this does not disable any enabled transition, denoted $\sigma \xrightarrow{t}_c \sigma'$, where τ is a special symbol. Notice that τ does not specify how much time passed, but it is used to label the time transitions.

Since our aim is to verify temporal (i.e., changing in time) properties of markings of a TPN $\mathcal{N} = (P, T, F, m^0, Eft, Lft)$, we use the propositional variables corresponding to its places. Formally, let $PV = \{\wp_p \backslash p \in P\}$ be a set of propositional variables, where the propositional variable $\wp_p$ corresponds to a place $p \in P$.

Definition 2 (Concrete Model for TPN) A *concrete model* for a time Petri net $\mathcal{N} = (P, T, F, m^0, Eft, Lft)$ is a tuple $M_c(\mathcal{N}) = ((\Sigma, \sigma^0, \rightarrow_c), V_c)$, where

- Σ is the set of all the concrete states of $\mathcal{N}$,
- σ^0 is the initial state,
- $\rightarrow_c$ is the transition relation, and
- $V_c \colon \Sigma \rightarrow PV$ is a *valuation function* such that $V_c((m, \cdot)) = \{\wp_p \mid p \in m\}$ i.e., V_c assigns the same propositions to concrete states with the same markings.

To abstract from the flow of time in the transition relation, in the definition of a concrete model instead of $\longrightarrow_c$ one can use also the *discrete transition relation* $\rightarrow_d \subseteq \Sigma \times T \times \Sigma$, defined as: $\sigma \xrightarrow{t}_d \sigma'$ iff $(\exists \sigma_1, \sigma_2 \in \Sigma)\sigma \xrightarrow{\tau^*}_c \sigma_1 \xrightarrow{t}_c \sigma_2 \xrightarrow{\tau^*}_c \sigma'$.

Unfortunately, concrete models are typically infinite. Therefore, we need to abstract them into preferably finite abstract ones, i.e., models whose nodes are sets of concrete states. The transitions of the abstract models are labeled with elements of the set $B = \mathrm{T} \cup \{\tau\}$, consisting of the transitions T of $\mathcal{N}$ and the special symbol τ, as defined below.

Definition 3 (Abstract Model for TPN) A structure $M_a(\mathcal{N}) = ((W, w^0, \rightarrow), V)$ is an *abstract model* for a concrete model $M_C(\mathcal{N}) = ((S, s^0, \rightarrow), V_c)$, where

- each node $w \in W$ is a set of states of S and $s^0 \in w^0$,
- $V(w) = V_c(s)$ for each $s \in w$,
- $\rightarrow \subset W \times B \times W$ such that

$$EE\colon\ (\forall w_1, w_2 \in W)\,(\forall b \in B) : w_1 \xrightarrow{b} w_2 \text{ if } (\exists s_1 \in \mathrm{w}_1)(\exists s_2 \in w_2) \text{ s.t. } \mathrm{s}_1 \xrightarrow{b} s_2.$$

The condition EE is illustrated in Fig. 2.

Definition 3 is very general, so it needs to be refined in order to ensure that abstract models preserve the properties expressible in a given temporal logic. Moreover Definition 3 does not give any clue how to build abstract models. We elaborate on these two issues in the next two sections.

2.2 Abstract Models Preserving Temporal Logics

Properties of timed systems are usually expressed using temporal logics. In this section our focus is on the logics that are most commonly, used, i.e., Linear Time Temporal Logic (LTL) and Computation Tree Logic* (CTL*); see Chapter 4 of [10]. LTL is interpreted over all the state sequences of a model starting at the initial state, while CTL* is interpreted over the tree resulting from unwinding a model starting

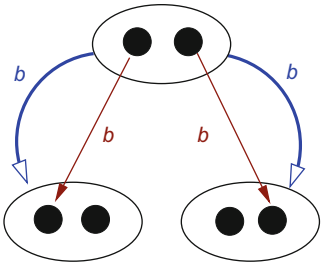

Fig. 2 An example of an abstract model satisfying the condition EE. The ovals represent the abstract states, the black dots stand for the concrete states, and the red (blue) arrows represent the concrete (abstract, resp.) transitions. Valuations of the nodes are omitted. The same graphical conventions apply to Figs. 3, 4, and 5

from the initial state. The formulas of LTL and of CTL* are built from propositional variables of PV using standard Boolean operators $\neg$, v, $\wedge$, and the state operators X (neXt), U (Until), and R (Release), the meanings of which correspond to their names. Since CTL* is interpreted over trees, its language contains also the two path quantifiers A (for All sequences) and E (there Exists a sequence) that allow us (to apply the state operators to some sequence or to all sequences starting at some state. We also use the restriction of CTL*, called ACTL*, such that negation can only by applied to the propositional variables and the operator E cannot be used. For example, $X\{\wp_p U \wp_q)$ is an LTL formula, $AG(EX(\wp_p U \wp_q))$ is a formula of CTL*, while $AG(AX(\wp_p U \wp_q))$ is an ACTL* formula.

Three more properties, EA, AE, and U, of abstract models are defined below in order to ensure the preservation of the three temporal logics, LTL, CTL*, and ACTL*, respectively. *Surjective models* are abstract models that satisfy the condition EA, given below.

These models preserve the logic LTL.

$$\text{EA}\colon\ (\forall w_1, w_2 \in W)\,(\forall b \in B) : w_1 \xrightarrow{b} w_2 \Rightarrow (\forall s_2 \in w_2)\,(\exists s_1 \in w_1) s_1 \xrightarrow{b} s_2.$$

The condition EA is illustrated in Fig. 3.

Bisimulating models are abstract models that satisfy the condition AE, defined below. These models preserve the logic CTL*.

$$\text{AE}\colon\ (\forall w_1, w_2 \in W)\,(\forall b \in B) : w_1 \xrightarrow{b} w_2 \Rightarrow (\forall s_1 \in w_1)\,(\exists s_2 \in w_2) s_1 \xrightarrow{b} s_2.$$

The condition AE is illustrated in Fig. 4.

Simulating models are abstract models that satisfy the condition U, defined below. These models preserve the logic ACTL*.

U: For each $w \in W$ there is a nonempty $w^{cor} \subseteq w$ such that $s^0 \in (w^0)^{cor}$, and

$$(\forall w_1, w_2 \in W)\,(\forall b \in B) : w_1 \xrightarrow{b} w_2 \Rightarrow \left(\forall s_1 \in {w_1}^{cor}\right)\left(\exists s_2 \in {w_2}^{cor}\right) s_1 \xrightarrow{b} s_2.$$

The condition U is illustrated in Fig. 5, where the blue ovals denote subsets w^{cor} of the abstract states and the blue-dashed-violet arrows represent the abstract transitions.

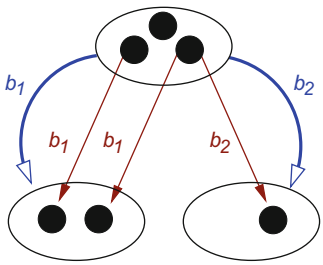

Fig. 3 An example of an abstract model satisfying the condition EA

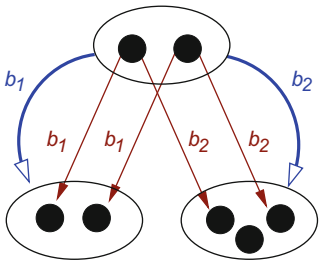

Fig. 4 An example of an abstract model satisfying the condition AE

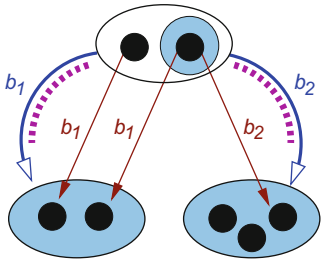

Fig. 5 An example of an abstract model satisfying the condition U

There are different methods of generating abstract models, descriptions of which can be found in [10]. One of the main methods, called *partitioning refinement*, starts with a small abstract model preserving only the propositional variables of a given formula. Then, this abstract model is refined by splitting states until one of the properties EA, AE, or U begins to hold, depending on which temporal logic the formula belongs to. Abstract models constructed this way are finite, thus they allow the standard model-checking method to be applied.

2.3 Partial Order Reductions (POR) [8]

However, it may turn out that the abstract models defined above are still too large to be efficiently verified. Then, we can weaken languages of temporal logics by disallowing the use of the next-step operator. This is indicated with $-X$ added to the name of each logic. Then, the equivalence to be preserved by a reduced model is weakened as well. Such an equivalence relation preserving CTL$^*-$X (ACTL$^*-$X, LTL$-$X) is called *stuttering bisimulation (simulation, trace equivalence, respectively)*.

The intuitive idea behind partial order reductions consists in generating reduced abstract models such that for each maximal path p in the abstract model, the reduced one contains at least one (but as few as possible) maximal path p', which differs from p only in the ordering of independent transitions. This independence relation is symmetric and is defined for Petri nets as follows. Two transitions t, t' are independent iff $(\bullet t \cup t\bullet) \cap (\bullet t' \cup t'\bullet) = \emptyset$, i.e., t and t' have no common input and no common output places. In the case of time Petri nets one can define an asymmetric covering relation by t covers t' iff t, t' are independent and $Eft(t) = 0$.

There are many algorithms that use this independency or covering relation to generate a reduced model without building the full one. At each node w of the generated reduced state space, the algorithm computes a subset (called *stubborn* or *ample*) of the set of all the enabled transitions and generates the successor nodes for the transitions of this subset only.

2.4 Symbolic Model Checking for TPN

At the Institute of Computer Science, Polish Academy of Sciences we have developed several new methods for verification of (parametric) time Petri nets. We considered two symbolic approaches, SAT- and BDD-based, to bounded model checking (BMC) of (parametric) time Petri nets and focused on the properties expressed in Linear Temporal Logic and the existential fragment of Computation Tree Logic. More specifically, we developed a SAT-based (parametric) reachability method for a class of distributed time Petri nets [11], which are TPNs composed of sequential processes, BMC approaches for verification of distributed time Petri nets [7], bounded parametric model checking for Petri nets [4], and a BDD-based BMC method for temporal properties of 1-Safe Petri nets [6]. Moreover, we proved and exploited the fact that the discrete-time semantics is sufficient to verify the properties formulated in the existential and the universal version of CTL* of time Petri nets with the dense semantics [2]. Recently, an SMT-based reachability-checking method for bounded time Petri nets was defined [13].

2.5 Verics

Our tool Verics [3] is a model checker composed of several independent modules aimed at verification of (parametric) time Petri nets, timed automata, and multiagent systems. The verification engine is mainly based on translations of the model-checking problem into the SAT problem. Depending on the type of considered system, the verifier enables us to test various classes of properties—from reachability of a state satisfying certain conditions to more complicated features expressed with formulas of (timed) temporal, epistemic, or deontic logics. The implemented model-checking methods include SAT-based ones as well as these based on generating abstract models for systems.

2.6 SAT-Based Bounded Model Checking

Bounded model checking is a symbolic method aimed at verification of temporal properties of distributed (timed) systems. It is based on the observation that some

properties of a system can be checked using only a part of its model. In order to apply SAT-based BMC to testing whether a system satisfies a certain (usually undesired) property, the transition relation of a given system is unfolded up to some depth k and encoded as a propositional formula. The formula expressing the property of interest is encoded as a propositional formula as well and satisfiability of the conjunction of these two formulas is checked using a SAT-solver. If the conjunction is satisfiable, one can conclude that a witness was found. Otherwise, the value of k is incremented. The above process is terminated when the value of k is equal to the diameter of the system, i.e., to the maximal length of a shortest path between its two arbitrary states.

2.7 Encoding for PN and TPN

Below we show how to encode the paths of length k. A set of global states is represented by a single *symbolic state*, i.e., as a vector of propositional variables w (called a *state variable vector*). Then, $k+1$ state variable vectors stand for a symbolic k-path, where the first symbolic state encodes the initial state of the system while the last one corresponds to the last states of the k-paths. The formula encoding the symbolic k-path is defined as follows:

$$path_k\left(\mathbf{w}^0, \ldots, \mathbf{w}^k\right) = \mathfrak{I}\left(\mathbf{w}^0\right) \wedge \bigwedge_{i=0}^{k-1} \mathfrak{T}\left(\mathbf{w}^i, \mathbf{w}^{i+1}\right) \tag{1}$$

where $\mathfrak{I}(\mathbf{w}^0)$ encodes the initial state of the system, and $\mathfrak{T}(\mathbf{w}^i, \mathbf{w}^{i+1})$ encodes a transition between symbolic states represented by the global state vectors $\mathbf{w}^i$ and $\mathbf{w}^{i+1}$.

In what follows we briefly describe how to define $\mathfrak{T}(\mathbf{w}^i, \mathbf{w}^{i+1})$ for PN and TPN.

Implementation for PN Consider a Petri net $\mathcal{N} = (P, T, F, m^0)$, where the places are denoted by the integers smaller than or equal to $n = |P|$. We use the set $\{p_1, \ldots, p_n\}$ of propositions, where p_i is interpreted as the presence of a token in the place i. The states of S are encoded by valuations of a vector of state variables $\mathbf{w} = (\mathbf{w}[1], \ldots, \mathbf{w}[n])$, where $\mathbf{w}[i] = p_i$ for $0 \leq i \leq n$. The initial state and the transition relation $\rightarrow$ are encoded as follows:

- $\mathfrak{I}\left(\mathbf{w}^0\right) := \bigwedge_{i \in m^0} \mathbf{w}^0[i] \wedge \bigwedge_{i \in P \setminus m^0} \neg \mathbf{w}^0[i]$,
- $\mathfrak{T}\left(\mathbf{w}, \mathbf{v}\right) := \bigvee_{t \in T} \Big(\bigwedge_{i \in {}^\bullet t} \mathbf{w}\left[i\right] \wedge \bigwedge_{i \in (t^\bullet \setminus pre(t))} \neg \mathbf{w}\left[i\right] \wedge \bigwedge_{i \in ({}^\bullet t \setminus t^\bullet)} \neg \mathbf{v}\left[i\right] \wedge \bigwedge_{i \in t^\bullet} \mathbf{v}\left[i\right] \wedge \bigwedge_{i \in (p \setminus ({}^\bullet t \cup t^\bullet)) \cup ({}^\bullet t \cap t^\bullet)} \mathbf{w}\left[i\right] \Leftrightarrow \mathbf{v}\left[i\right]\Big)$.

A more detailed description and the encoding of the formulas of the existential fragment of CTL can be found in [11].

Implementation for Distributed TPN The main difference between the symbolic encoding of the transition relation of PN and TPN consists in the time flow. Below, we give some details of the encoding for distributed TPN. A current state of a TPN $\mathcal{N}$ is given by its marking and the time passed since each of the enabled transitions became enabled (which influences the future behavior of the net). Thus, a *concrete state* σ of $\mathcal{N}$ can be defined as an ordered pair $(m, clock)$, where m is a marking and $clock\colon : I \to \mathbb{R}_+$ is a function, which for each index i of a process of $\mathcal{M}$ gives the time elapsed since the marked place of this process became marked most recently [12]. In order to deal with countable structures instead of uncountable ones, we can use *extended detailed region graphs,* i.e., abstract models based on the well-known detailed region graphs introduced by Alur et al. for timed automata [1].

To apply the BMC approach we deal with a model obtained by a *discretization* of its extended detailed region graph. The model is of an infinite but countable structure, which, however, is sufficient for BMC (which deals with finite sequences of states only). Instead of dealing with the whole extended detailed region graph, we *discretize* this structure, choosing for each region one or more appropriate representatives. The discretization scheme is based on the one for timed automata [14], and it preserves the qualitative behavior of the underlying system. The details and the formal definitions can be found in [11].

2.8 Bounded Parametric Model Checking

BMC is also applied to verification of properties expressed in the existential fragment of the logic PRTCTL, which is an extension of Computation Tree Logic (CTL) by allowing the formulation of properties involving lengths of paths in a model. For example, consider $\mathrm{EG}^{\leq 5}\wp$, which expresses the fact that there is a path such that in the first six states of this path $\wp$ holds. Another example is $\forall_{\Theta_1 \leq 1}\exists_{\Theta_2 \leq 2}\mathrm{EF}^{\leq \Theta_1+\Theta_2}\wp$, which expresses the fact that for each value of the parameter Θ_1 not greater than 1 there is a value of the parameter Θ_2 not exceeding 2 such that $\wp$ can be reached at a prefix of length at most $\Theta_1 + \Theta_2 + 1$ of some path. The propositions appearing in these formulas correspond to the places of the net considered. In order to apply verification using BMC, the qualitative properties expressed in PRTECTL are directly encoded as propositional formulas.

3 Final Remarks

We have discussed some features of verification that are specific to Petri nets, in particular model abstraction techniques, partial order reductions, and SAT-based bounded-model-checking methods for (time) Petri nets.

Verification of (time) Petri nets is a very active area of research with many new verification methods emerging every year. This is motivated by broad practical applications of time Petri nets to model concurrent systems, real-time systems, stochastic systems, and hybrid systems. It is also important to mention that the efficiency of tools for (time) Petri nets is improving every year (see http://mcc.lip6.fr/).

Acknowledgements The author is grateful to Dr. Agata Półrola for reading and commenting on this paper, which is mainly based on the joint book [10] and joint research.

References

1. R. Alur, C. Courcoubetis, D. Dill, Model checking in dense real-time. Inf. Comput. **104**(1), 2–34 (1993)
2. A. Janowska, W. Penczek, A. Półrola, A. Zbrzezny, Using integer time steps for checking branching time properties of time Petri nets. in *Trans. Petri Nets and Other Models of Concurrency*, ed. by M. Koutny, W.M.P. van der Aalst, A. Yakovlev. LNCS, vol. 8100(8) (Springer, Berlin, 2013), pp. 89–105
3. M. Knapik, A. Niewiadomski, W. Penczek, A. Półrola, M. Szreter, A. Zbrzezny, Parametric model checking with VerICS, in *Trans. Petri Nets and Other Models of Concurrency*, ed. by M. Knapik et al. LNCS, vol. 6550(4) (Springer, Berlin, 2010), pp. 98–120
4. M. Knapik, M. Szreter, W. Penczek, Bounded parametric model checking for elementary net systems, in *Trans. Petri Nets and Other Models of Concurrency*, ed. by M. Knapik et al. LNCS, vol. 6550(4), 42–71 (Springer, Berlin, 2010)
5. A. Mazurkiewicz. Trab theory, in *Advances in Petri Nets 1986*, ed. by W. Braner, W. Reisig, G. Rozenberg. LNCS, vol. 255 (Springer, Berlin, 1986), pp. 279–324
6. A. Męski, W. Penczek, A. Półrola, BDD-based bounded model checking for temporal properties of 1-safe Petri nets. Fundam. Inform. **109**(3), 305–321 (2011)
7. A. Męski, A. Półrola, W. Penczek, B. Woźzna-Szcześniak, A. Zbrzezny, Bounded model checking approaches for verification of distributed time Petri nets, in *Proceedings of the International Workshop on Petri Nets and Software Engineering* (*PNSE'11*) (2011), pp. 72–91
8. W. Penczek, A. Półrola, Abstractions and partial order reductions for checking branching properties of time Petri nets, in *Proceedings of the 22nd International Conference on Applications and Theory of Petri Nets* (*ICATPN'01*), ed. by J.M. Colom, M. Koutny. LNCS, vol. 2075 (Springer, Berlin, 2001), pp. 323–342
9. W. Penczek, A. Półrola, Specification and model checking of temporal properties in time Petri nets and timed automata, in *Proceedings of the 25th International Conference on Applications and Theory of Petri Nets (ICATPN '04)*, ed. by J. Cortadella, W. Reisig. LNCS, vol. 3099 (Springer, Berlin, 2004), pp. 37–76
10. W. Penczek, A. Półrola, *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*. Studies in Computational Intelligence, vol. 20 (Springer, Berlin, 2006)
11. W. Penczek, A. Półrola, A. Zbrzezny, SAT-based (parametric) reachability for a class of distributed time Petri nets, in *Trans. Petri Nets and Other Models of Concurrency*, ed. by M. Knapik et al. LNCS, vol. 6550(4) (Springer, Berlin, 2010), pp. 72–97

12. A. Półrola, W. Penczek, Minimization algorithms for time Petri nets. Fundam. Inform. **60**(1–4), 307–331 (2004)
13. A. Półrola, P. Cybula, A. Meski, SMT-based reachability checking for bounded time Petri nets. Fundam. Inform. **135**(4), 467–882 (2014)
14. A. Zbrzezny, SAT-based reachability checking for timed automata with diagonal constraints. Fundam. Inform. **67**(1–3), 303–322 (2005)