Specification and Model Checking of Temporal Properties in Time Petri Nets and Timed Automata*

Wojciech Penczek[1,2] and Agata Półrola[3]

[1] Institute of Computer Science, PAS, Ordona 21, 01-237 Warsaw, Poland
[2] Institute of Informatics, Podlasie Academy, Sienkiewicza 51, 08-110 Siedlce, Poland
penczek@ipipan.waw.pl
[3] Faculty of Mathematics, University of Lodz, Banacha 22, 90-238 Lodz, Poland
polrola@math.uni.lodz.pl

Abstract. The paper surveys some of the most recent approaches to verification of properties, expressible in some timed and untimed temporal logics (LTL, CTL, TCTL), for real-time systems represented by time Petri nets (TPN's) and timed automata (TA). Firstly, various structural translations from TPN's to TA are discussed. Secondly, model abstraction methods, based on state class approaches for TPN's, and on partition refinement for TA, are given. Next, SAT-based verification techniques, like bounded and unbounded model checking, are discussed. The main focus is on bounded model checking for TCTL and for reachability properties. The paper ends with a comparison of experimental results for several time Petri nets, obtained using the above solutions, i.e., either model abstractions for TPN's, or a translation of a net to a timed automaton and then verification methods for TA. The experiments have been performed using some available tools for TA and TPN's.

1 Introduction

Model checking provides for a promising set of techniques for hardware and software verification. Essentially, in this formalism verifying that a property follows from a system specification amounts to checking whether or not a temporal formula is valid on a model representing all possible computations of the system.

Recently, the interest in automated verification is moving towards concurrent real-time systems. Several models of such systems are usually considered in the literature, but timed automata (TA) [6] and time Petri nets (TPN's) [51] belong to the most widely used. For the above systems, one is, usually, interested in checking reachability or temporal properties that are most frequently expressed in either a standard temporal logic like LTL and CTL*, or in a timed extension of CTL, called TCTL [3].

* Partly supported by the State Committee for Scientific Research under the grant No. 3T11C 00426 and a special grant supporting ALFEBIITE

However, practical applicability of model checking methods is strongly limited by the *state explosion problem*. For real-time systems, the problem occurs with a particular strength, which follows from infinity of the time domain. Therefore, existing verification techniques frequently apply symbolic representations of state spaces using either operations on Difference Bound Matrices [34], variations of Boolean Decision Diagrams [11, 84, 84], or SAT-related algorithms. The latter can exploit either a sequence of translations starting from timed automata and TCTL, going via (quantified) separation logic to quantified propositional logic and further to propositional logic [10, 57, 73] or a direct translation from timed automata and TCTL to propositional logic [63, 86, 92]. Finite state spaces, preserving properties to be checked, are usually built using detailed region approach or (possibly minimal) abstract models based on state classes or regions. Algorithms for generating such models have been defined for time Petri nets [14, 17, 35, 48, 58, 61, 82, 89], as well as for timed automata [3, 4, 21, 32, 65, 79].

It seems that in spite of the same underlying timed structure, model checking methods for time Petri nets and timed automata have been defined independently of each other. However, several attempts to combine the two approaches have been already made, concerning both a structural translation of one model to the other [28, 37, 42, 49, 64, 74] or an adaptation of existing verification methods [35, 58, 82].

The goal of this paper is to report on a recent progress in building abstract state spaces and application of symbolic methods for verification of both the time Petri nets and timed automata, either directly or indirectly via a translation from the former formalism to the latter. To this aim, firstly, various structural translations from TPN's to TA are discussed (see Section 3) to apply timed automata specific methods to time Petri nets. Temporal specification languages are introduced in Section 4. Model abstraction methods based on state classes approaches for TPN's and on partition refinement for TA are given in Section 5. Next, SAT-based verification techniques, like bounded (BMC) and unbounded model checking (UMC), are discussed in Section 6.

The idea behind BMC consists in translating the model checking problem for an existential fragment of some temporal logic (like ECTL or TECTL) on a fraction of a model into a test of propositional satisfiability, for which refined tools already exist [56]. Unlike BMC, UMC [50, 73] deals with unrestricted temporal logics checked on complete models at the price of a decrease of efficiency. In this paper the main focus is on SAT-related methods of BMC for TCTL [63] as well as for reachability and unreachability [86].

Each section of our paper is accompanied with pointers to the literature, where descriptions of complementary verification methods can be found. The paper ends with a comparison of experimental results for several time Petri nets, obtained using solutions discussed in the paper, i.e., either model abstraction techniques for TPN's, or a translation of a net to a timed automaton and then verification methods for TA. The experiments, described in Section 8, have been performed using the tools: Tina [16], Kronos [91], and VerICS [31].

2 Main Models of Real-Time Systems

We consider two main models of real-time systems: Petri nets with time and timed automata. First we define Petri nets, discuss their time extensions, and provide a definition of time Petri nets. Our attention is focused on a special kind of TPN's - distributed time Petri nets, which are then considered in our case studies.

The following abbreviations are used in definitions of both TA and TPN's. Let $\mathbb{R}$ ($\mathbb{R}_+$) denote the set of non-negative (positive) reals, $\mathbb{N}$ ($\mathbb{N}_+$) - the set of (positive) naturals, and $\mathbb{Q}_+$ - the set of non-negative rational numbers,

2.1 Petri Nets with Time

We start with the standard notion of Petri nets.

Definition 1. *A Petri net is a four-element tuple* $\mathcal{P} = (P, T, F, m_0)$*, where* $P = \{p_1, \ldots, p_{n_P}\}$ *is a finite set of* places*,* $T = \{t_1, \ldots, t_{n_T}\}$ *is a finite set of* transitions*,* $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ *is the* flow function*, and* $m_0 : P \to \mathbb{N}$ *is the* initial marking *of* $\mathcal{P}$*.*

For each transition $t \in T$ we define its *preset* $\bullet t = \{p \in P \mid F(p,t) > 0\}$ and its *postset* $t\bullet = \{p \in P \mid F(t,p) > 0\}$. Moreover, in order to simplify some consequent notions, we consider only the nets, for which $\bullet t$ and $t\bullet$ are non-empty, for all transitions t. We use the following auxiliary notations and definitions:

- a *marking* of $\mathcal{P}$ is any function $m : P \to \mathbb{N}$;
- a transition $t \in T$ is *enabled* at m ($m[t\rangle$ for short) if $(\forall p \in \bullet t)\ m(p) \geq F(p,t)$, and *leads from marking* m *to* m', where for each $p \in P$, $m'(p) = m(p) - F(p,t) + F(t,p)$. The marking m' is denoted by $m[t\rangle$ as well, if this does not lead to misunderstanding;
- $en(m) = \{t \in T \mid m[t\rangle\}$ - the set of transitions enabled at m;
- for $t \in en(m)$, $newly\_en(m,t) = \{u \in T \mid u \in en(m[t\rangle) \ \wedge\ u \notin en(m')$ with $m'(p) = m(p) - F(p,t)$ for each $p \in P\}$ - the set of transitions newly enabled after firing t;
- a marking m is *reachable* if there exists a sequence of transitions $t^1, \ldots, t^l \in T$ and a sequence of markings $m^0, \ldots, m^l$ such that $m^0 = m_0$, $m^l = m$, and $t^i \in en(m^{i-1})$, $m^i = m^{i-1}[t^i\rangle$ for each $i \in \{1, \ldots, l\}$; the set of all the reachable markings of $\mathcal{P}$ is denoted by $RM_{\mathcal{P}}$;
- a net $\mathcal{P}$ is said to be *bounded* if all its reachable markings are bounded;
- two transitions $t_1, t_2 \in T$ are *concurrently enabled in* m if $t_1 \in en(m)$ and $t_2 \in en(m')$ with $m'(p) = m(p) - F(p,t_1)$ for each $p \in P$;
- a net $\mathcal{P}$ is *sequential* if none of its reachable markings concurrently enables two transitions;
- a net $\mathcal{P}$ is *ordinary* if the flow function F maps onto $\{0,1\}$;
- a net $\mathcal{P}$ is *1-safe* if it is ordinary and $m(p) \leq 1$, for each $p \in P$ and each $m \in RM_{\mathcal{P}}$.

Intuitively, Petri nets are directed weighted graphs with two types of nodes: places (representing conditions) and transitions (representing events), whose arcs correspond to these elements in the domain of the flow function, for which the value of this function is positive. The arcs are assigned positive weights according to the values of F.

The theory of Petri nets provides a general framework for modelling distributed and concurrent systems. Since for many of them timing dependencies play an important role, a variety of extensions of the main formalism, enabling to reason about temporal properties, has been introduced. In what follows, we present a brief survey of the approaches, based on [23, 71].

Petri nets with timing dependencies can be classified according to the way of specifying timing constraints (these can be timing intervals [51, 83] or single numbers [68]), or elements of the net with which these constraints are associated (places [27], transitions [51, 68] or arcs [1, 38, 83]). The next criterion is the interpretation of the timing constraints. When associated with a *transition*, the constraint can be viewed as its *firing time* (a transition consumes the input tokens when becomes enabled, but does not create the output ones until the delay time associated with it has elapsed [68]), *holding time* (when the transition fires, the actions of removing and creating tokens are done instantaneously, but the tokens created are not available to enable new transitions until they have been in their output place for the time specified as the duration time of the transition which created them [81]), or *enabling time* (a transition is forced to be enabled for a specified period of time before it can fire, and tokens are removed and created in the same instant [51]). A time associated with a *place* usually refers to the period the tokens must spend in the place before becoming available to enable a transition [27]. A timing interval on an *input arc* usually expresses the conditions under which tokens can potentially leave the place using this arc [38], whereas a timing interval on an *output arc* denotes the time when tokens produced on this arc become available [83]. Nets can be also classified according to *firing rules*: the *weak firing rule* means that the time which passes between enabling of the transition and its firing is not determined [80], the *strong earliest firing rule* requires the transition to be fired as soon as it is enabled and the appropriate timing conditions are met [38], whereas the *strong latest firing rule* means that the transition can be fired in a specified period of time, but no later than after certain time from its enabling, unless it becomes disabled by firing of another (conflicting) one [51]. The best known timed extensions of Petri nets are *timed Petri nets* by Ramchandani [68] and *time Petri nets* by Merlin and Farber [51]. In this paper, we focus on the latter.

Definition 2. *A time Petri net (TPN, for short) is a six-element tuple* $\mathcal{N} = (P, T, F, m_0, Eft, Lft)$*, where* (P, T, F, m_0) *is a Petri net, and* $Eft : T \to \mathbb{N}$*,* $Lft : T \to \mathbb{N} \cup \{\infty\}$ *are functions describing the* earliest *and the* latest firing times *of the transitions, where* $Eft(t) \leq Lft(t)$ *for each* $t \in T$*.*

A *concrete state* σ of $\mathcal{N}$ is an ordered pair $(m, clock)$, where m is a marking, and $clock : T \to \mathbb{R}$ is a function which for each transition $t \in en(m)$ gives

the time elapsed since t became enabled most recently. By $clock + \delta$ we denote the function given by $(clock + \delta)(t) = clock(t) + \delta$ for all $t \in T$. Moreover, let $(m, clock) + \delta$ denote $(m, clock + \delta)$. The *concrete state space* of $\mathcal{N}$ is a structure $F_c(\mathcal{N}) = (\Sigma, \sigma^0, \rightarrow)$, where Σ is the set of all the concrete states of $\mathcal{N}$, $\sigma^0 = (m_0, clock_0)$ with $clock_0(t) = 0$ for each $t \in T$ is the initial state, and a timed consecution relation $\rightarrow \subseteq \Sigma \times (T \cup \mathbb{R}) \times \Sigma$ is defined by action- and time-successors as follows:

- for $\delta \in \mathbb{R}$, $(m, clock) \xrightarrow{\delta} (m, clock + \delta)$ iff $(clock + \delta)(t) \leq Lft(t)$ for all $t \in en(m)$ (*time successor*),
- for $t \in T$, $(m, clock) \xrightarrow{t} (m_1, clock_1)$, iff $t \in en(m)$, $Eft(t) \leq clock(t) \leq Lft(t)$, $m_1 = m[t\rangle$, and for all $u \in T$ we have $clock_1(u) = 0$ for $u \in newly\_en(m, t)$, and $clock_1(u) = clock(u)$ otherwise (*action successor*).

A σ_0-*run* ρ of $\mathcal{N}$ is a maximal sequence of concrete states $\rho = \sigma_0 \xrightarrow{\delta_0} \sigma_0 + \delta_0 \xrightarrow{t_0} \sigma_1 \xrightarrow{\delta_1} \sigma_1 + \delta_1 \xrightarrow{t_1} \sigma_2 \xrightarrow{\delta_2} \ldots$, where $t_i \in T$ and $\delta_i \in \mathbb{R}$, for all $i \in \mathbb{N}$. A state $\sigma \in \Sigma$ is *reachable* if there exists a σ^0-run ρ and $i \in \mathbb{N}$ such that $\sigma = \sigma_i + \delta_i$.

The set of all the reachable states of $\mathcal{N}$ is denoted by $Reach_{\mathcal{N}}$. A marking m is reachable if there is a state $(m, clock) \in Reach_{\mathcal{N}}$ for some function $clock$. A run ρ is said to be *progressive* iff $\Sigma_{i \in \mathbb{N}} \delta_i$ is unbounded.

The above notion of run can be used for interpreting untimed branching time temporal logics [58] or for checking reachability. Alternatively, runs can be defined such that each two consecutive time and action steps are combined [89].[1] However, in order to give semantics over dense paths for timed temporal logics, we assume that $\delta_i > 0$ for all $i \in \mathbb{N}$. This will become clear in Section 4.2.

In what follows, we consider only nets whose all the runs are infinite[2] and progressive by restricting the nets to contain neither two consecutive transitions whose earliest firing times are both equal to 0 nor one such a transition, which is a loop[3]. The set of all the σ-runs of $\mathcal{N}$, with $\delta_i > 0$ for all $i \in \mathbb{N}$, is denoted by $f_{\mathcal{N}}(\sigma)$.

In order to reason about systems represented by TPN's, we define, for a given set of propositional variables PV, a valuation function $V_{\mathcal{N}} : P \rightarrow 2^{PV}$, which assigns propositions to the places[4] of $\mathcal{N}$. Let $V_c : \Sigma \rightarrow 2^{PV}$ be a valuation function extending $V_{\mathcal{N}}$ such that $V_c((m, \cdot)) = \bigcup_{p \in m} V_{\mathcal{N}}(p)$, i.e., V_c assigns the same propositions to the states with the same markings. The structure $M_c(\mathcal{N}) = (F_c(\mathcal{N}), V_c)$ is called a *concrete model* of $\mathcal{N}$. Notice that for the same concrete model, we can interpret timed and untimed logics using appropriate notions of runs.

A time Petri net $\mathcal{N} = (P, T, F, m_0, Eft, Lft)$ is said to be sequential if the net $\mathcal{P} = (P, T, F, m_0)$ is so, and $\mathcal{N}$ is *1-safe* if for each reachable marking m of $\mathcal{N}$, we have $m(p) \leq 1$ for each $p \in P$. Unless otherwise stated, in what follows

[1] This semantics is claimed to be equivalent w.r.t. CTL model checking.

[2] This can be checked by applying algorithms looking for deadlocks.

[3] This restriction is only for assuring an existence of a translation to timed automata, for which similar restrictions are assumed.

[4] Usually, there is one-to-one correspondence between the propositions and the places.

1-safe TPN's are considered only. Moreover, we define a notion of a *distributed time Petri net*, which is an adaptation of the one from [41].

Definition 3. *Let I be a finite set of indices, and let $\mathfrak{N} = \{N^i \mid i \in I\}$, with $N^i = (P^i, T^i, F^i, Eft^i, Lft^i, m_0^i)$ be a family of 1-safe, sequential time Petri nets (called* processes*), indexed I, with pairwise disjoint sets P^i of places, and satisfying the condition $(\forall i_1, i_2 \in I)(\forall t \in T^{i_1} \cap T^{i_2})\ (Eft^{i_1}(t) = Eft^{i_2}(t) \wedge Lft^{i_1}(t) = Lft^{i_2}(t))$. A* distributed time Petri net *$\mathcal{N} = (P, T, F, Eft, Lft, m_0)$ is the union of the* processes *N^i, i.e., $P = \bigcup_{i\in I} P^i$, $T = \bigcup_{i\in I} T^i$, $F = \bigcup_{i\in I} F^i$, $Eft = \bigcup_{i\in I} Eft^i$, $Lft = \bigcup_{i\in I} Lft^i$ and $m_0 = \bigcup_{i\in I} m_0^i$.*

It is easy to notice that a distributed net is 1-safe. The interpretation of such a net is a collection of sequential, non-deterministic processes with communication capabilities (via joint transitions). In what follows, we consider distributed nets whose all the processes are *state machines* (i.e., for each $t \in T^i$, $|\bullet t| = |t\bullet| = 1$), which implies that in any reachable marking m of $\mathcal{N}$, there is exactly one place p of each process with $m(p) = 1$. It is important to mention that each distributed net can be transformed to an equivalent net satisfying this requirement [43].

Example 1. Examples of (distributed) time Petri net are shown in Fig. 1. Each of the nets 5*a*, 5*b*, and 5*c* in the left-hand side consists of two disjoint processes with the sets of places $P^1 = \{p_1, p_2, p_3, p_4\}$ and $P^2 = \{p_6, p_7\}$, whereas the net on the right is composed of three communicating processes with the sets of places: $P^i = \{idle_i, trying_i, enter_i, critical_i\}$ with $i = 1, 2$, and $P^3 = \{place0, place1, place2\}$.

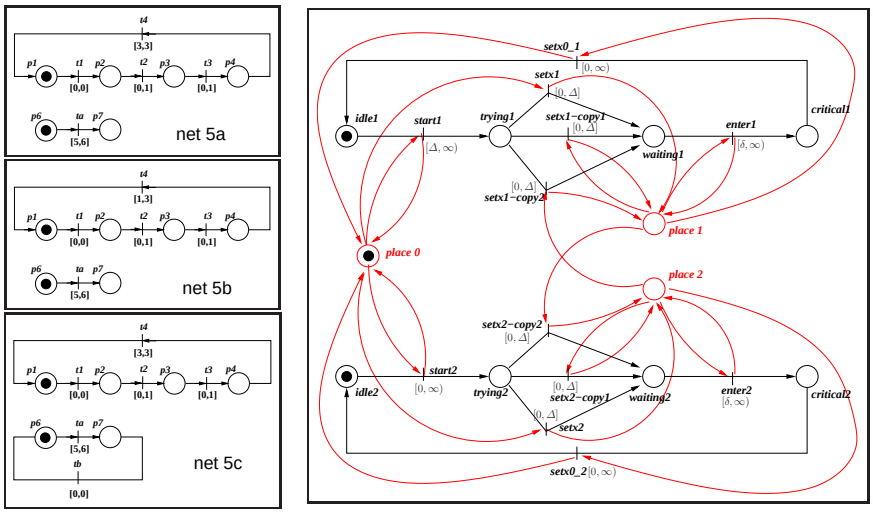


Fig. 1. The nets of [89] and a net for Fischer's mutual exclusion protocol

2.2 Timed Automata

In this section we briefly recall the concept of timed automata, which was introduced by Alur and Dill [6]. Timed automata are extensions of finite state automata with constraints on timing behaviour. The underlying finite state automata are augmented with a set of real time variables. We start with formalizing the above notions. Let $\mathcal{X} = \{x_1, \ldots, x_{n_\mathcal{X}}\}$ be a finite set of real-valued variables, called *clocks*. The set of *clock constraints* over $\mathcal{X}$ is defined by the following grammar:

$$\mathfrak{cc} := true \mid x_i \sim c \mid x_i - x_j \sim c \mid \mathfrak{cc} \wedge \mathfrak{cc},$$

where $x_i, x_j \in \mathcal{X}$, $c \in \mathbb{N}$, and $\sim \in \{\leq, <, =, >, \geq\}$. The set of all the clock constraints over $\mathcal{X}$ is denoted by $\mathcal{C}^{\ominus}_{\mathcal{X}}$, whereas its restriction, where differences of clocks are not allowed, is denoted by $\mathcal{C}_{\mathcal{X}}$. A *clock valuation* on $\mathcal{X}$ is a $n_\mathcal{X}$-tuple $v \in \mathbb{R}^{n_\mathcal{X}}$. For simplicity, we assume a fixed ordering on $\mathcal{X}$. The value of the clock x_i in v can be then denoted by $v(x_i)$ or $v(i)$, depending on the context. For a valuation v and $\delta \in \mathbb{R}$, $v + \delta$ denotes the valuation v' s.t. for all $x \in \mathcal{X}$, $v'(x) = v(x) + \delta$. Moreover, for a subset of clocks $X \subseteq \mathcal{X}$, $v[X := 0]$ denotes the valuation v' such that for all $x \in X$, $v'(x) = 0$ and for all $x \in \mathcal{X} \setminus X$, $v'(x) = v(x)$. Let $v \in \mathbb{R}^{n_\mathcal{X}}$, the satisfaction relation $\models$ for a clock constraint $\mathfrak{cc} \in \mathcal{C}^{\ominus}_{\mathcal{X}}$ is defined inductively as follows:

- $v \models true$,
- $v \models (x_i \sim c)$ iff $v(x_i) \sim c$,
- $v \models (x_i - x_j \sim c)$ iff $v(x_i) - v(x_j) \sim c$, and
- $v \models (\mathfrak{cc} \wedge \mathfrak{cc}')$ iff $v \models \mathfrak{cc}$ and $v \models \mathfrak{cc}'$.

For a constraint $\mathfrak{cc} \in \mathcal{C}^{\ominus}_{\mathcal{X}}$, let $[\![\mathfrak{cc}]\!]$ denote the set of all the clock valuations satisfying $\mathfrak{cc}$, i.e., $[\![\mathfrak{cc}]\!] = \{v \in \mathbb{R}^{n_\mathcal{X}} \mid v \models \mathfrak{cc}\}$. By a (*time*) *zone* in $\mathbb{R}^{n_\mathcal{X}}$ we mean each convex polyhedron $Z \subseteq \mathbb{R}^{n_\mathcal{X}}$ defined by a clock constraint, i.e., $Z = [\![\mathfrak{cc}]\!]$ for some $\mathfrak{cc} \in \mathcal{C}^{\ominus}_{\mathcal{X}}$ (for simplicity, we identify the zones with the clock constraints which define them). The set of all the zones for $\mathcal{X}$ is denoted by $Z(n_\mathcal{X})$.

Given $v, v' \in \mathbb{R}^{n_\mathcal{X}}$ and $Z, Z' \in Z(n_\mathcal{X})$, we define the following operations:

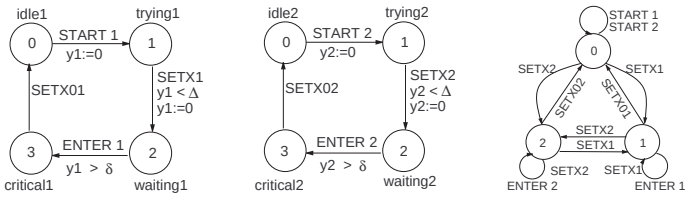
- $Z \setminus Z'$ is a set of disjoint zones s.t. $\{Z'\} \cup (Z \setminus Z')$ is a partition of Z;
- $Z \nearrow := \{v' \in \mathbb{R}^n \mid (\exists v \in Z)\ v \leq v'\}$, where $v \leq v'$ iff $\exists \delta \in \mathbb{R}$ s.t. $v' = v + \delta$;
- $Z[X := 0] = \{v[X := 0] \mid v \in Z\}$.

Notice that the operations $\nearrow$, $Z[X := 0]$, and the standard intersection preserve zones. These results and the implementation of $Z \setminus Z'$ can be found in [4, 79].

Definition 4. *A* timed automaton *(TA, for short) is a six-element tuple $\mathcal{A} = (A, L, l^0, E, \mathcal{X}, \mathcal{I})$, where A is a finite set of actions, L is a finite set of locations, $l^0 \in L$ is an initial location, $\mathcal{X}$ is a finite set of clocks, $E \subseteq L \times A \times \mathcal{C}^{\ominus}_{\mathcal{X}} \times 2^{\mathcal{X}} \times L$ is a transition relation. Each element e of E is denoted by $l \xrightarrow{a, \mathfrak{cc}, X} l'$, which represents a transition from the location l to the location l', executing the action a, with the set $X \subseteq \mathcal{X}$ of clocks to be reset, and with the clock constraint $\mathfrak{cc}$ defining the enabling condition for e. The function $\mathcal{I} : L \to \mathcal{C}^{\ominus}_{\mathcal{X}}$, called a* location invariant*, assigns to each location $l \in L$ a clock constraint defining the conditions under which $\mathcal{A}$ can stay in l.*

If the enabling conditions and the values of the location invariant are in the set $\mathcal{C}_{\mathcal{X}}$ only, then the automaton is called *diagonal-free*. In what follows, we consider diagonal-free automata. In order to reason about systems represented by timed automata, for a set of propositional variables PV, define a valuation function $V_{\mathcal{A}} : L \to 2^{PV}$, which assigns propositions to the locations. Usually, we consider networks of timed automata, where the automata are composed by synchronization over labels. The reader is referred to [86] for a formal definition of composition.

Example 2. In Fig. 2, a network of TA for Fischer's mutual exclusion protocol with two processes is depicted[5]. The protocol is parameterised by the number of processes involved. In the general case, the network consists of n automata of processes, together with one automaton modelling a global variable X, used to coordinate the processes' access to their critical sections, which means that if the automaton of a process and the automaton of the variable X contain actions with a common label a, then the process is allowed to execute this action if and only if the automaton for X executes an action labelled by a as well.



Process 1 Process 2 Variable X

Fig. 2. Fischer's Mutual Exclusion Protocol for two processes

A *concrete state* of $\mathcal{A}$ is a pair (l, v), where $l \in L$ and $v \in \mathbb{R}^{n_{\mathcal{X}}}$ is a valuation. The *concrete* (*dense*) *state space* of $\mathcal{A}$ is a structure $F_c(\mathcal{A}) = (Q, q^0, \to)$, where $Q = L \times \mathbb{R}^{n_{\mathcal{X}}}$ is the set of all the concrete states, $q^0 = (l^0, v^0)$ with $v^0(x) = 0$ for all $x \in \mathcal{X}$ is the initial state, and $\to \subseteq Q \times (A \cup \mathbb{R}) \times Q$ is the transition relation, defined by action- and time-successors as follows:

- for $a \in A$, $(l, v) \xrightarrow{a} (l', v')$ iff $(\exists \mathfrak{cc} \in \mathcal{C}_{\mathcal{X}})(\exists X \subseteq \mathcal{X})$ such that $l \xrightarrow{a,\mathfrak{cc},X} l' \in E$, $v \in [\![\mathfrak{cc}]\!]$, $v' = v[X := 0]$ and $v' \in [\![\mathcal{I}(l')]\!]$ (*action successor*),
- for $\delta \in \mathbb{R}$, $(l, v) \xrightarrow{\delta} (l, v + \delta)$ iff $v + \delta \in [\![\mathcal{I}(l)]\!]$ (*time successor*).

For $(l, v) \in Q$, let $(l, v) + \delta$ denote $(l, v + \delta)$. A *q-run* ρ of $\mathcal{A}$ is a maximal sequence of concrete states: $q_0 \xrightarrow{\delta_0} q_0 + \delta_0 \xrightarrow{a_0} q_1 \xrightarrow{\delta_1} q_1 + \delta_1 \xrightarrow{a_1} q_2 \xrightarrow{\delta_2} \ldots$, where $q_0 = q \in Q$, $a_i \in A$ and $\delta_i \in \mathbb{R}$, for each $i \geq \mathbb{N}$. A state $q' \in Q$ is *reachable* if there exists a q^0-run $\rho = q_0 \xrightarrow{\delta_0} q_0 + \delta_0 \xrightarrow{a_0} q_1 \xrightarrow{\delta_1} \ldots$ and $i \in \mathbb{N}$ such that

[5] Two clocks are denoted by y_1 and y_2, whereas Δ and δ are parameters.

$q' = q_i + \delta_i$. The set of all the reachable states of $\mathcal{A}$ will be denoted by $Reach_{\mathcal{A}}$. A run ρ is said to be *progressive* iff $\Sigma_{i \in \mathbb{N}} \delta_i$ is unbounded. A timed automaton is *progressive* iff all its runs are progressive. For simplicity of the presentation, we consider only progressive timed automata. Note that progressiveness can be checked using sufficient conditions of [79].

Like for time Petri nets, the above notion of run can be easily used for interpreting untimed temporal logics or for checking reachability. However, in order to give semantics over dense paths corresponding to runs (see Section 4.2) for timed temporal logics, we assume that $\delta_i > 0$ for all $i \in \mathbb{N}$ in a run. So, by $f_{\mathcal{A}}(q)$ we denote the set of all such q-runs of $\mathcal{A}$.

The structure $M_c(\mathcal{A}) = (F_c(\mathcal{A}), V_c)$ is called a *concrete model* for $\mathcal{A}$, for a valuation function $V_c : Q \to 2^{PV}$ extending $V_{\mathcal{A}}$ such that $V_c(l, v) = V_{\mathcal{A}}(l)$ (i.e., V_c assigns the same propositions to the states with the same locations).

3 From TPN's to TA

There are two approaches to verifying properties of time Petri nets. Either specific algorithms are used or nets are translated to timed automata in order to exploit verification methods designed for automata[6]. Therefore, we first consider translations from TPN's to TA, and then review the most recent verification methods for both the formalisms.

Several methods of translating time Petri nets to timed automata have been already developed. However, in most cases translations produce automata, which extend timed automata. Some of the existing approaches are sketched below.

The most natural translation consists in defining an automaton whose locations correspond to the markings of the net, whereas the actions and the clocks - to its transitions. The invariant of a marking m expresses that no transition can be disabled by passage of time, whereas the enabling condition of an action labelled by $t \in T$ guarantees that the time passed since t became enabled is between $Eft(t)$ and $Lft(t)$. Executing an action labelled by t at m resets the clocks corresponding to the transitions in $newly\_en(m, t)$. This approach was used to define *detailed region graphs* for TPN's [58, 82]. Its formal description can be also found in [64].

Sifakis and Yovine [74] presented a translation of a subclass of *time stream Petri nets* [72] to automata whose invariants can be disjunctions of clock constraints. In these nets, each of the arcs (p, t) is assigned a timing interval which specifies when tokens in the place p become available to fire the transition t. An enabled transition t can be fired if (*) all the places in $\bullet t$ have been marked at least as long as the lower bounds of the corresponding intervals, and (**) there is at least one place in $\bullet t$ marked no longer than the upper bound of the corresponding interval. (1-safe) time Petri nets can be seen as a subclass of these nets. In order to translate a net $\mathcal{N}$ to an automaton, we define a location for each marking of $\mathcal{N}$, and associate a clock with each of its places. The actions are

[6] Usually, the concrete state spaces of both the models are required to be bisimilar.

labelled by the transitions of $\mathcal{N}$. Executing an action labelled by $t \in T$ resets the clocks of the places in $t\bullet$, whereas its enabling condition corresponds to (*) given above. The invariant of a marking m states that (**) holds for each $t \in en(m)$.

In [25, 36], translations of (general) TPN's to automata equipped with shared variables and urgency modelling mechanisms are provided. The method of [25] generates a set of TA containing an automaton $\mathcal{A}_t$ with one clock for each $t \in T$, and a supervisor automaton $\mathcal{A}_s$. The locations of $\mathcal{A}_t$ correspond to the possible states of t (being enabled, disabled, and to its firing). The automaton $\mathcal{A}_s$ with *committed locations*[7] forces the other automata to change synchronously their states when a transition of the net is fired. Shared variables are used to model the number of tokens in the places. In the approach of [36], the transitions are classified according to their number of input, output and inhibitor places, and one automaton with the locations *disabled* and *enabled* is built for each of the classes obtained. Similarly as in [25], an additional automaton (with an *urgent transition*[8]) ensures a synchronous behaviour of the whole system when a transition is fired, whereas shared variables store the marking of the net.

Lime and Roux [49] proposed a method of translating a (general) time Petri net to a timed automaton, using an algorithm of building a *state class graph* G [14] (see Sec. 5.1). The nodes of G are *state classes*, i.e., pairs (m, I), where m is a marking and I is a set of inequalities. The translation leads to an automaton whose locations are the nodes of G, and the edges, labelled by transitions of the net, correspond to its successor relation. Every class (m, I) is assigned a set of clocks, each of which corresponds to all these transitions in $en(m)$ which became enabled at the same time. Executing an action labelled by t resets some clocks (associated with newly enabled transitions). It is also possible to assign a value of one clock to another (this goes beyond the standard definition of TA). Invariants and enabling conditions describe, respectively, when the net can be in a given marking, or when a transition can be fired at a class. Since the number of state classes is finite only if the net is bounded [14], the authors provide a condition for an on-line checking of unboundedness to ensure stopping the computation.

The paper [28] shows a method of translating extended 1-safe TPN's with finite values of the function Lft (called PRES+ models) to a set of extended timed automata. When applied to a non-extended net, the translation gives a network of standard TA. The set of transitions of the net is divided into disjoint *clusters*, i.e., sequences of transitions in which firing of one of them enables the next one. An automaton with one clock is built for each such a cluster.

Another translation [64] applies to distributed nets only. It is based on a different approach to the concrete semantics of the net, which is bisimulation equivalent[9] to that defined in Sec. 2.1. In this semantics, each process is assigned a "clock" measuring the time elapsed since its place became marked most recently. The automaton resulting from the translation uses the above clocks, whereas its locations are defined as the markings of the net, and its edges are

[7] A committed location is a location which has to be leaved as soon as it is entered.
[8] A transition which has to be taken as soon as it is enabled.
[9] The concrete state spaces are bisimilar, see [64] for a proof.

labelled by the transitions. Firing of a transition resets the clocks related to the processes involved. The enabling conditions and invariants are possibly disjunctions of clock constraints, but can be modified to be zones (see [64] for details).

Besides translations from TPN's to TA, some methods of translating subclasses of TA to time Petri nets also exist [37], but because of lack of space, we do not discuss these approaches in this paper.

4 Main Formalisms for Expressing Properties

Properties of timed systems are usually expressed using temporal logics. In this section we look at the logics that are most commonly used. We start with untimed formalisms, which are later extended with time constraints.

4.1 Untimed Temporal Logics: LTL, CTL, CTL*

All the untimed logics we consider are subsets of CTL*. Therefore, we give syntax and semantics for CTL* only. The other logics are defined as its restrictions.

Syntax and Semantics of CTL*. Let $PV_L = \{\wp_1, \wp_2 \ldots\}$ be a set of propositional variables. The language of CTL* is given as the set of all the state formulas φ_s, defined using path formulas φ_p, by the following grammar:

$$\varphi_s := \wp \mid \neg\wp \mid \varphi_s \wedge \varphi_s \mid \varphi_s \vee \varphi_s \mid \mathrm{A}\varphi_p \mid \mathrm{E}\varphi_p$$
$$\varphi_p := \varphi_s \mid \varphi_p \wedge \varphi_p \mid \varphi_p \vee \varphi_p \mid \mathrm{X}\varphi_p \mid \varphi_p \mathrm{U} \varphi_p \mid \varphi_p \mathrm{R} \varphi_p$$

A ('for all paths') i E ('there exists a path') are path quantifiers, whereas X ('neXt'), U ('Until'), and R ('Release') are state operators. The formula $\varphi_p \mathrm{U} \psi_p$ expresses that ψ_p eventually occurs and that φ_p holds continuously until then. The operator R is dual to U. Derived path operators are: $\mathrm{G}\varphi_p \stackrel{def}{=} (\wp \wedge \neg\wp)\mathrm{R}\varphi_p$ and $\mathrm{F}\varphi_p \stackrel{def}{=} (\wp \vee \neg\wp)\mathrm{U}\varphi_p$. Sublogics of CTL* are defined below:

- CTL (Computation Tree Logic): the temporal formulas are restricted to positive boolean combinations of: $\mathrm{A}(\varphi \mathrm{U} \psi), \mathrm{A}(\varphi \mathrm{R} \psi)$, $\mathrm{AX}\varphi$, and $\mathrm{E}(\varphi \mathrm{U} \psi), \mathrm{E}(\varphi \mathrm{R} \psi)$, $\mathrm{EX}\varphi$ only.
- ACTL (Universal Computation Tree Logic): the temporal formulas are restricted to positive boolean combinations of: $\mathrm{A}(\varphi \mathrm{U} \psi), \mathrm{A}(\varphi \mathrm{R} \psi)$, and $\mathrm{AX}\varphi$ only.
- ECTL (Existential Computation Tree Logic): the temporal formulas are restricted to positive boolean combinations of: $\mathrm{E}(\varphi \mathrm{U} \psi), \mathrm{E}(\varphi \mathrm{R} \psi)$, and $\mathrm{EX}\varphi$ only.
- LTL (Linear Time Logic): the formulas of the form $\mathrm{A}\varphi$ are allowed only, where φ does not contain the path quantifiers A, E.
- $\mathbf{L}_{-\mathrm{X}}$ denotes logic L without the next step operator X.

For example, $\mathrm{AFG}(\wp_1 \vee \wp_2)$ is an LTL formula, whereas $\mathrm{AFAG}(\wp_1 \vee \wp_2)$ is an ACTL formula. Each of the above logics can be extended by time constraints (defined later). Semantics of CTL* is defined over standard Kripke models.

Definition 5. *A* model *is a tuple* $M = ((S, s^0, \rightarrow), V)$*, where* S *is a set of states,* $s^0 \in S$ *is the initial state,* $\rightarrow \subseteq S \times S$ *is a total successor relation, and* $V : S \rightarrow 2^{PV}$ *is a valuation function.*

Let $|M|$ denote the number of states of M. For $s_0 \in S$ a path $\pi = (s_0, s_1, \ldots)$ is an infinite sequence of states in S starting at s_0, where $s_i \rightarrow s_{i+1}$ for all $i \geq 0$, and $\pi_i = (s_i, s_{i+1}, \ldots)$ is the i-th suffix of π.

- $M, s \models \wp$ iff $\wp \in V(s)$, $M, s \models \neg\wp$ iff $\wp \notin V(s)$,
- $M, x \models \varphi \vee \psi$ iff $M, x \models \varphi$ or $M, x \models \psi$, for $x \in \{s, \pi\}$,
- $M, x \models \varphi \wedge \psi$ iff $M, x \models \varphi$ and $M, x \models \psi$, for $x \in \{s, \pi\}$,
- $M, s \models \mathrm{A}\varphi$ iff $M, \pi \models \varphi$ for each path π starting at s,
- $M, s \models \mathrm{E}\varphi$ iff $M, \pi \models \varphi$ for some path π starting at s,
- $M, \pi \models \varphi$ iff $M, s_0 \models \varphi$, for a state formula φ,
- $M, \pi \models \mathrm{X}\varphi$ iff $M, \pi_1 \models \varphi$,
- $M, \pi \models \psi \mathrm{U} \varphi$ iff $(\exists j \geq 0)$ $\big(M, \pi_j \models \varphi$ and $(\forall 0 \leq i < j)$, $M, \pi_i \models \psi\big)$,
- $M, \pi \models \psi \mathrm{R} \varphi$ iff $(\forall j \geq 0)$ $\big(M, \pi_j \models \varphi$ or $(\exists 0 \leq i < j)$, $M, \pi_i \models \psi\big)$.

We adopt the initialized notion of validity in a model: $M \models \varphi$ iff $M, s^0 \models \varphi$.

For timed systems, untimed temporal logics are typically interpreted over concrete (or abstract) models, where a path is defined to correspond to a run that includes unrestricted time-successor steps (i.e., with $\delta_i \in \mathbb{R}$). The *model checking* problem for CTL* over timed systems is defined as follows: given a CTL* formula φ and a timed system $\mathcal{T}$ (i.e., a TPN $\mathcal{N}$ or a TA $\mathcal{A}$) together with a valuation function $V_{\mathcal{T}}$, determine whether $M_c(\mathcal{T}) \models \varphi$.

Besides untimed properties of timed systems, which are directly expressed using the above-defined temporal logics, *reachability* in these systems is usually checked. Given a propositional formula p, the reachability problem for timed automata (time Petri nets) consists in testing whether there is a reachable state satisfying p in $M_c(\mathcal{A})$ ($M_c(\mathcal{N})$, resp.). This problem can be obviously translated to the model checking problem for the CTL* formula EFp. However, in spite of that, several efficient solutions aimed at reachability checking only exist as well.

4.2 Timed Temporal Logics

Timed temporal logics can be interpreted over either discrete or dense models of time [9]. We consider the latter option. Since the model checking problem for TCTL* is undecidable [2], we focus on TCTL [3] and its subsets: TACTL and TECTL, defined analogously to the corresponding fragments of CTL.

The logic TCTL is an extension of CTL$_{-\mathrm{X}}$ obtained by subscribing the modalities with time intervals specifying time restrictions on formulas. Formally, syntax of TCTL is defined inductively by the following grammar:

$$\varphi := \wp \mid \neg\wp \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathrm{A}(\varphi \mathrm{U}_I \varphi) \mid \mathrm{E}(\varphi \mathrm{U}_I \varphi) \mid \mathrm{A}(\varphi \mathrm{R}_I \varphi) \mid \mathrm{E}(\varphi \mathrm{R}_I \varphi)$$

where $\wp \in PV_L$ and I is an interval in $\mathbb{R}$ with integer bounds of the form $[n, n']$, $[n, n')$, $(n, n']$, (n, n'), (n, ∞), and $[n, \infty)$, for $n, n' \in \mathbb{N}$.

For example, $\mathrm{A}((\wp \wedge \neg\wp)\mathrm{R}_{[0,\infty)}(\wp_1 \Rightarrow \mathrm{A}((\wp \vee \neg\wp)\ \mathrm{U}_{[0,5]}\ \wp_2)))$ expresses that for all the runs, always when $\wp_1$ holds, $\wp_2$ holds within 5 units of time.

Semantics of TCTL over TA and TPN's. Let $\mathcal{T}$ be a timed system, i.e., a TPN $\mathcal{N}$ or a TA $\mathcal{A}$, $M_c(\mathcal{T}) = (F_c(\mathcal{T}), V_c)$ be its concrete model, and $\rho = s_0 \xrightarrow{\delta_0} s_0 + \delta_0 \xrightarrow{b_0} s_1 \xrightarrow{\delta_1} s_1 + \delta_1 \xrightarrow{b_1} s_2 \xrightarrow{\delta_2} \dots$ be an s_0-run of $\mathcal{T}$, where $\delta_i \in \mathbb{R}_+$ for $i \in \mathbb{N}$. Recall that $f_{\mathcal{T}}(s)$, for a concrete state s, denotes the set of all the progressive runs of $\mathcal{T}$ starting at s (where the time steps are labelled with $\delta > 0$ only). In order to interpret TCTL formulas along a run, we introduce the notion of a *dense path corresponding to* ρ, denoted by π_ρ, which is a mapping from $\mathbb{R}$ to a set of states[10], given by $\pi_\rho(r) = s_i + \delta$ for $r = \Sigma_{j=0}^{i}\delta_j + \delta$, with $i \geq 0$ and $0 \leq \delta < \delta_i$. Next, we define semantics of TCTL formulas in the following way:

$s_0 \models \wp$ iff $\wp \in V_c(s_0)$,
$s_0 \models \neg\wp$ iff $\wp \notin V_c(s_0)$,
$s_0 \models \varphi \vee \psi$ iff $s_0 \models \varphi$ or $s_0 \models \psi$,
$s_0 \models \varphi \wedge \psi$ iff $s_0 \models \varphi$ and $s_0 \models \psi$,
$s_0 \models \mathrm{A}(\varphi \mathrm{U}_I \psi)$ iff $(\forall\, \rho \in f_{\mathcal{T}}(s_0))(\exists r \in I)\big[\pi_\rho(r) \models \psi \wedge (\forall r' < r)\ \pi_\rho(r') \models \varphi\big]$,
$s_0 \models \mathrm{E}(\varphi \mathrm{U}_I \psi)$ iff $(\exists\, \rho \in f_{\mathcal{T}}(s_0))(\exists r \in I)\big[\pi_\rho(r) \models \psi \wedge (\forall r' < r)\ \pi_\rho(r') \models \varphi\big]$,
$s_0 \models \mathrm{A}(\varphi \mathrm{R}_I \psi)$ iff $(\forall\, \rho \in f_{\mathcal{T}}(s_0))(\forall r \in I)\big[\pi_\rho(r) \models \psi \vee (\exists r' < r)\ \pi_\rho(r') \models \varphi\big]$,
$s_0 \models \mathrm{E}(\varphi \mathrm{R}_I \psi)$ iff $(\exists\, \rho \in f_{\mathcal{T}}(s_0))(\forall r \in I)\big[\pi_\rho(r) \models \psi \vee (\exists r' < r)\ \pi_\rho(r') \models \varphi\big]$.

Again, we adopt the initialized notion of validity in a model: $M_c(\mathcal{T}) \models \varphi$ iff $M_c(\mathcal{T}), s^0 \models \varphi$, where s^0 is the initial state in $M_c(\mathcal{T})$.

It is important to mention that there is an alternative semantics for sublogics of $\mathrm{CTL}_{-\mathrm{X}}$, which can be interpreted over dense models by assuming that semantics of each untimed modality O is like semantics of the corresponding TCTL modality $\mathrm{O}_{[0,\infty)}$. The *model checking* problem for TCTL over a timed system $\mathcal{T}$ is defined as usual, i.e., given a TCTL formula φ and a timed system together with a valuation function $V_{\mathcal{T}}$, determine whether $M_c(\mathcal{T}) \models \varphi$.

5 Verification Methods for TPN's and TA

Basic methods of verification of timed systems consist in building their finite (possibly minimal) *abstract models* that preserve properties of interest, and then in running verification algorithms on these models, for instance, a version of the *state-labelling algorithm* of [3], or an algorithm which checks an equivalence of behaviours between the model and its specification.

Let $M_c(\mathcal{T}) = ((S, s^0, \to), V_c)$ be a concrete model of a timed system $\mathcal{T}$, where $\to$ is a B-labelled transition relation for a given set of labels B. By an abstract model we mean a structure $M_a = ((W, w^0, \to), V)$, where elements of W, called *abstract states*, are sets of concrete states of S, and satisfy at least the following conditions: $s^0 \in w^0$; $V(w) = V_c(s)$ for each $w \in W$ and $s \in w$, and for any $w_1, w_2 \in W$ and each $b \in B$

EE) $w_1 \xrightarrow{b} w_2$ if there are $s_1 \in w_1$ and $s_2 \in w_2$ s.t. $s_1 \xrightarrow{b} s_2$.

Condition EE guarantees that abstract states are related if they contain related representatives. Other conditions on $\to$ depend on the properties to be preserved. Some of them are listed below:

[10] This can be defined thanks to the assumption about $\delta > 0$.

EA) $w_1 \xrightarrow{b} w_2$ iff $(\exists s_1 \in w_1)(\forall s_2 \in w_2)\ s_1 \xrightarrow{b} s_2$;
AE) $w_1 \xrightarrow{b} w_2$ iff $(\forall s_1 \in w_1)(\exists s_2 \in w_2)\ s_1 \xrightarrow{b} s_2$;
U) $w_1 \xrightarrow{b} w_2$ iff $(\forall s_1 \in w_1^{cor})(\exists s_2 \in w_2^{cor})\ s_1 \xrightarrow{b} s_2$,
where $w^{cor} \subseteq w$ for each $w \in W$, and $s^0 \in (w^0)^{cor}$.

Condition EA restricts the abstract transition relation to the pairs of states, where all the representatives of the successor state have a related representative in the predecessor state. This condition is put on abstract models to preserve LTL. Condition AE, which specifies the symmetric property w.r.t. the successor and the predecessor of each pair of states in the abstract transition relation, is known as a *bisimulation condition*. So, it is used to ensure preservation of CTL* or TCTL. Condition U is a weakening of AE, which puts the same restriction, but only on a subset of each abstract state. It is know as a *simulation condition*. Similarly, U is applied to preserve ACTL* or TACTL. Next, we present the main approaches to generating various kinds of abstract models for timed systems.

5.1 State Classes Approaches

Methods of building abstract models for time Petri nets are usually based on the *state classes approach*, which consists in representing an abstract state by a marking and a set of linear inequalities. Many algorithms for building such models, for various restrictions on the definition of TPN's and approaches to their concrete semantics, are known in the literature. Below, we present the main solutions. For each of them, we give either the method for obtaining the model from some earlier-defined underlying structure, or a full description, i.e., a definition of the concrete states applied, a notion of a state class, and a condition on firability of a transition at a class together with a method of computing its successor resulting from this firing. Moreover, in order to obtain a finite abstract model, an equivalence relation on state classes is provided. In all the descriptions, we use the following notation: given a set of inequalities I, $sol(I)$ denotes the set of its solutions, and $var(I)$ - the set of all the variables appearing in I.

State class graph [14, 15]. The method of building this basic model of the state classes approach was defined for general TPN's. Their concrete states are represented by pairs $\sigma^F = (m, f)$, where m is a marking, and f is a *firing interval function* which assigns to each $t \in en(m)$ the timing interval in which t is (individually) allowed to fire. A *state class* of a TPN is a pair $C = (m, I)$, where m is a marking, and I is a set of inequalities built over the set $var(I) = \{v_i \mid t_i \in en(m)\}$. All the possible values of a variable v_i in the set $sol(I)$ (called the *firing domain* of C) form the timing interval, relative to the time C was entered, in which t_i can be fired. The state class is thus a set of concrete states whose values of firing interval functions are included in the firing domain.

The initial class of the graph is given by $(m_0, \{\text{“}Eft(t_i) \leq v_i \leq Lft(t_i)\text{“} \mid t_i \in en(m_0)\})$. A transition $t_i \in en(m)$ is *firable* at a class $C = (m, I)$ if the set of inequalities $I_1 = I \cup \{\text{“}v_i \leq v_j\text{“} \mid t_j \in en(m)\}$ is consistent (i.e., $sol(I_1) \neq \emptyset$), which intuitively means that t_i can fire earlier than any other enabled transition.

The class $C' = (m', I')$, resulting from firing t_i, satisfies $m' = m[t_i\rangle$, whereas I' is obtained from I by the following four steps: (1) the set I is substituted by I_1 (i.e., the firablity condition for t_i is added to I), (2) all the variables in I are substituted by new ones reflecting the fact of firing t_i at the time given by v_i which relates the values of the variables to the time the class C' was entered, (3) the variables corresponding to the transitions disabled by firing of t_i are eliminated, and (4) the system is extended by the set of inequalities $\{\text{“}Eft(t_l) \leq v_l \leq Lft(t_l)\text{“} \mid t_l \in newly\_en(m,t)\}$.

Two classes are considered as equivalent if their markings and firing domains are equal. A TPN is of a bounded number of state classes if and only if it is bounded. Although the boundedness problem for TPN's is undecidable, in [14, 15] sufficient conditions for checking unboundedness while generating the state class graph are given. Since all the state classes satisfy the condition EA, the model preserves LTL formulas.

Geometric region graph [89]. The information carried by firing domains of the classes of the state class graph is not sufficient to check their *atomicity*[11]. To this aim, some additional information about the histories of firings is needed. Therefore, as a step towards building a model preserving CTL* properties, a modification of the state class graph for 1-safe nets with finite values of the function Lft, called *geometric region graph*, has been introduced. Its construction exploits the notion of concrete states given in Sec. 2.1. State classes are triples $C = (m, I, \eta)$, where I is a set of inequalities, and m is a marking obtained by firing from m_0 the sequence $\eta \in T^*$ of transitions, satisfying the timing constraints represented by I. The variables in $var(I)$ represent absolute (i.e., counted since the net started) firing times of the transitions in η (different firings of the same transition are then distinguished, and $v_i^j \in var(I)$ corresponds to j-th firing of $t_i \in T$). Unlike the construction of [14, 15], I can be seen as describing the history of states of a given class rather than these states as such.

The initial state class is given by $C^0 = (m_0, \emptyset, \epsilon)$, where ϵ is the empty sequence of transitions. Firing of $t_i \in en(m)$ at a class $C = (m, I, \eta)$ is described in terms of the *parents* of the enabled transitions, i.e., the transitions is η which most recently made the transitions in $en(m)$ enabled[12]. If t_i appears $k-1$ times in η, then it is firable at C iff the set of inequalities $I_1 = I \cup \{\text{“}parent(v_i^k, C) + Eft(t) \leq parent(v_j^l, C) + Lft(t_j)\text{“} \mid t_j \in en(m)$ and t_j appears $l-1$ times in $\eta\}$ is consistent[13], which means that t_i can be fired earlier than any other enabled transition. In the class $C' = (m, I', \eta')$, obtained by firing t_i at C, $m' = m[t\rangle$, $\eta = \eta t$, and the set of inequalities I' is equal to $I_1 \cup \{\text{“}Eft(t_i) \leq t_i^k - parent(t_i^k, C) \leq Lft(t_i)\text{“}\}$. This describes timing conditions which need to hold for firing of t_i at C. The classes are equivalent if their markings are equal,

[11] A class which satisfies the condition AE w.r.t. all its successors is called *atomic*.

[12] As the parents of the transitions enabled in m_0, a fictitious transition ν, which denotes the start of the net and fires at the time 0, is assumed.

[13] $parent(v_i, C)$ denotes the variable corresponding to the most recent appearance of the parent of t_i in η.

the enabled transitions have the same parents, and these parents could be fired at the same absolute times. All the classes of the graph satisfy the condition EA.

Atomic state class graph [89]. In order to build a model which preserves CTL* properties, the geometric state class graph needs to be refined until all its classes are atomic. The model obtained this way is called *atomic state class graph*. If a class $C = (m, I, \eta)$ is not atomic, then there is some inequality ξ non-redundant[14] in I and such that satisfaction of ξ is necessary for the concrete states in C to have descendants in a successor C' of C. The class C is then split into $C_1 = (m, I \cup \xi, \eta)$ and $C_2 = (m, I \cup \neg\xi, \eta)$. Their descendants are computed from copies of those of C, by modifying their sets of inequalities adding ξ and $\neg\xi$, respectively. As a result, the abstract states satisfy both AE and EA.

Pseudo-atomic state class graph [61]. The atomic state class graph's construction was further modified to generate *pseudo-atomic class graphs* which preserve the universal fragment of CTL* (i.e., ACTL*). Instead of AE, all the classes of these graphs satisfy the weaker condition U. The models are built in a way similar to atomic state class graphs, besides the fact that in some cases instead of splitting the classes only their *cor*s are refined.

Strong state class graph [17]. The paper by Berthomieu and Vernadat presents another approach to building models which are then refined to a CTL*-preserving structure, applicable[15] to the general class of TPN's. The solution combines, in some sense, these of [89] and [14, 15]. The definition of concrete states is taken from [14]. The authors define a *strong state class graph*, whose classes are of the form $C = (m, I)$, where I is a set of inequalities built over the set of variables corresponding to the transitions in $en(m)$, similarly as in [14, 15]. However, the value of a variable v_i corresponding to a transition $t_i \in en(m)$ gives the time elapsed since t_i was last enabled, which, in turn, corresponds in some sense to Yoneda's approach. The set I enables to compute a firing domain, and in consequence to define the set of concrete states that belong to the class.

The initial class of the graph is given by $C^0 = (m_0, \{\text{“}0 \leq v_i \leq 0\text{“} \mid t_i \in en(m_0)\})$. Firability of a transition t at a class as well as the set of inequalities of the successor are defined in terms of the times elapsed since the transitions were enabled, using additional temporary variables denoting possible firing times of t (see [17] for details). Two classes (m, I) and (m', I') are considered as equivalent if $m = m'$ and $sol(I) = sol(I')$. Similarly as in the previous approaches, all the classes in the strong state class graph satisfy the condition EA.

Strong atomic state class graph [17]. In order to obtain a *strong atomic state class graph* preserving CTL* properties, the above model is refined in a way similar to the one of [89], i.e., its classes are partitioned until all of them are atomic. However, unlike [89], the classes satisfy AE but not necessarily EA, since an inequality added to the set I of a class C, which is partitioned, is not propagated to the descendants of C.

[14] A inequality ξ is non-redundant in the set of inequalities I if the solution sets of both $I \cup \xi$ and $I \cup \neg\xi$ are non-empty.

[15] However, transitions with infinite latest firing times require a special treatment.

The graphs of [48, 90]. In [90], a construction of a state class graph, applicable to 1-safe nets with finite values of the function Lft, was described. The method aimed at verification of formulas of a logic TNL (*Timed Temporal Logic for Nets*). Since building of these graphs is formula-guided, we do not provide it here, but focus on a similar approach proposed by Lilius for 1-safe nets with possibly infinite values of Lft, aimed at reachability checking using partial order reductions[16] [48]. It is based on the notion of concrete states given in Sec. 2.1. A state class is a pair (m, I), where m is a marking, and I is a set of inequalities describing the constraints on possible firing times of the transitions in $en(m)$. Values of the variables in $var(I)$ represent absolute times. There are two kinds of variables in $var(I)$: these which correspond to possible firing times of transitions in $en(m)$, and those which represent firing times of some of the transitions that are disabled in m and have been fired earlier. The latter are kept to determine the latest possible firing time of these $t \in en(m)$ which can become enabled by firings of different transitions. The set of inequalities of the initial class expresses that the differences between the possible firing times of each $t \in en(m_0)$ and the start of the net are between $Eft(t)$ and $Lft(t)$. Given a class (m, I), the set of inequalities of its successor $(m[t\rangle, I')$ obtained by firing $t \in en(m)$ is computed similarly as in Yoneda's approach, besides the fact that the condition saying that t can be fired earlier than any other $t' \in en(m)$ is not added to I' (see [90] and [48] for a detailed description and comparison). As a result, the solutions of I do not necessarily determine a feasible clock assignment as it was in [90]. However, there is a run of the net corresponding to each path in the graph. Applying an equivalence of classes ensures finiteness of the model, but, unlike the case of [90], it is not of a form of a tree. This is useful for partial order reductions.

The reachability graph of [22]. A completely different approach to state classes, aimed at computing reachability graphs of (general) time Petri nets, was proposed in [22]. It is based on a notion of *firing points* of transitions (defined as the instant when a given transition fires). The class produced by the $(n-1)$-th firing point (the points are numbered along sequences of transitions fired from m_0) is a triple (m, TS, TE), where m is a marking, $TS : \mathbb{N} \to en(m)$ gives the enabling order of transitions (the transitions of $TS(0)$ become enabled before those of $TS(1)$ etc.), whereas $TE : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}_+ \cup \{\infty\}$ is a function which for a given pair i, j returns the minimal or the maximal time elapsed between the i-th and the j-th firing point at the point $n-1$. The resulting graph is finite iff the net is bounded.

5.2 Detailed Region Graph Approach

Next, we define abstract models for timed automata, which can be used for enumerative and symbolic verification. We start with a detailed region graph approach. The main reason for introducing this approach is that our SAT-related methods (see Section 6) are based on a propositional encoding of detailed regions.

[16] Firing transitions in different orders can lead to the same state.

Given a timed automaton $\mathcal{A}$ and a TCTL formula φ. Let $\mathcal{C}_\mathcal{A} \subseteq \mathcal{C}_\mathcal{X}$ be a non-empty set containing all the clock constrains occurring in any enabling condition or in a state invariant of $\mathcal{A}$. Moreover, let $c_{max}(\varphi)$ be the largest constant appearing in $\mathcal{C}_\mathcal{A}$ and in any time interval in φ[17]. For $\delta \in \mathbb{R}$, $frac(\delta)$ denotes the fractional part of δ, and $\lfloor \delta \rfloor$ denotes its integral part.

Definition 6 (Equivalence of clock valuations). *For two clock valuations $v, v' \in \mathbb{R}^{n_\mathcal{X}}$, $v \simeq_{\mathcal{C}_{\mathcal{A},\varphi}} v'$ iff for all $x, x' \in \mathcal{X}$ the following conditions are met:*

1. *$v(x) > c_{max}(\varphi)$ iff $v'(x) > c_{max}(\varphi)$,*
2. *if $v(x) \leq c_{max}(\varphi)$ and $v(x') \leq c_{max}(\varphi)$ then*
   a.) *$\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$,*
   b.) *$frac(v(x)) = 0$ iff $frac(v'(x)) = 0$, and*
   c.) *$frac(v(x)) \leq frac(v(x'))$ iff $frac(v'(x)) \leq frac(v'(x'))$.*

Lemma 1 (Preserving TCTL, [3]).
Let $\mathcal{A} = (A, L, l^0, E, \mathcal{X}, \mathcal{I})$ be a timed automaton, $V_\mathcal{A}$ be a valuation function for $\mathcal{A}$, and $M_c(\mathcal{A})$ be the concrete model of $\mathcal{A}$. Moreover, let $l \in L$, and $v, v' \in \mathbb{R}^{n_\mathcal{X}}$ with $v \simeq_{\mathcal{C}_{\mathcal{A},\varphi}} v'$. Then, we have $M_c(\mathcal{A}), (l, v) \models \varphi$ iff $M_c(\mathcal{A}), (l, v') \models \varphi$.

The equivalence classes of the relation $\simeq_{\mathcal{C}_{\mathcal{A},\varphi}}$ are called *detailed zones*. The set of all the detailed zones is denoted by $DZ(n_\mathcal{X})$. A *detailed region* is a pair (l, Z), where $l \in L$ and $Z \in DZ(n_\mathcal{X})$.

The action and time successor relation in the set of the detailed regions can be defined via representatives (see [3, 63]), which gives us a finite abstract model, called the *detailed region graph* (DRG) preserving TCTL. In Section 6.2, we show how to encode detailed regions together with the transition relation in a symbolic way to accomplish bounded model checking.

5.3 Partition Refinement for TA

Partition refinement (minimization) is an algorithmic method of constructing abstract models for timed automata *on-the-fly*, i.e., without building concrete models first. Typically, the algorithm starts from an initial *partition* Π_0 of the state space Q of $\mathcal{A}$ (i.e., a set of disjoint *classes* the union of which equals Q), which respects (at least) the propositions of interest. The partition is then successively refined until it (or its reachable part, depending on the algorithm) becomes *stable*, i.e., satisfies the conditions required on the model to be generated. Let $M = (G, V)$ with $G = (W, w^0, \rightarrow)$ be a model generated by the above algorithm, where the elements of W are (reachable) classes of the stable partition Π, w^0 is the class containing the state q^0 of $\mathcal{A}$, and the successor relation is induced by that on Π. The classes of partitions are usually represented by regions, defined as follows. Given a timed automaton $\mathcal{A}$, let $l \in L$ and $Z \in Z(n_\mathcal{X})$. A *region* $R \subseteq L \times \mathbb{R}^{n_\mathcal{X}}$ is a set of concrete states $R = \{(l, v) \mid v \in Z\}$, denoted by (l, Z). For regions $R = (l, Z)$ and $R' = (l, Z')$, we define their difference

[17] Obviously, ∞ is not considered as a constant.

$R \setminus R' = \{(l, Z'') \mid Z'' \in Z \setminus Z'\}$. This operation potentially returns a set of regions, and is of exponential complexity in the number of clocks, which can cause some inefficiency of partition refinement algorithms.

Bisimulating models. Three main minimization algorithms were introduced in [20, 59, 47]. They are aimed at generating *bisimulating models*, i.e., models whose classes satisfy the condition AE. The set of labels B usually corresponds to the set of actions of the automaton augmented with one additional label τ which denotes passing some time[18].

An adaptation of the method of [47] to the case of timed automata was presented in [87]. The algorithm starts from an initial partition Π_0 which respects the invariants and enabling conditions of $\mathcal{A}$ (i.e., for each $Y \in \Pi$, each $e = l' \stackrel{a,\mathfrak{cc},X}{\longrightarrow} l'' \in E$ and each $l \in L$ we have $Y \cap [\![\mathcal{I}(l)]\!], Y \cap [\![\mathfrak{cc}]\!] \in \{Y, \emptyset\}$), and stabilizes only the reachable classes. To ensure this, each reachable class is *marked* by a representative that is guaranteed to be a reachable state of $\mathcal{A}$ (initially, only the initial class is marked by q^0). If there exists a transition from a representative of the reachable class Y_1 to a state in a class Y_2, then Y_2 becomes reachable, and is marked by one of its known reachable states. If Y_1 is not stable, then it is partitioned into a class Y_1' (easily computed using the inverse images of the successors), which contains a representative r_{Y_1} and all the concrete states of Y_1 which have their successors in the same classes as r_{Y_1}, and $Y_1 \setminus Y_1'$ (which possibly is a set of classes). The concrete states known as reachable in $Y_1 \setminus Y_1'$ (if exist) are chosen as the representatives of (parts of) $Y_1 \setminus Y_1'$. Partitioning of Y_1 can make its predecessors unstable. The paper proposes also a solution to the problem of computing differences of regions. It is based on using a *forest structure*, consisting of trees corresponding to the locations of $\mathcal{A}$, which keep the history of partitionings.

The algorithm of [20] was applied to timed automata in many papers, and served to building bisimulating models for various time-abstracted (*ta*-) bisimulation relations. These are *strong ta-bisimulation* abstracting away the exact amount of time passed between two states, and *delay* and *observational* ta-bisimulations, which additionally consider as equivalent the states obtained by executing an action a and these obtained by passing some time and then executing a, or, respectively, passing some time, executing a, and then passing some time again (see [79] for details). Its generic pseudo-code is presented in Fig. 3. The algorithm starts from an initial class containing the state q^0, and then successively searches and refines reachable classes. A class $Y_1 \in \Pi$ which is unstable w.r.t. its successor Y_2 is split into Y_1' which contains all the concrete states which have successors in Y_2, and $Y_1 \setminus Y_1'$ (this can possibly be a set of regions). Partitioning of a class can result in unstability of its predecessors.

In [5], the algorithm of Fig. 3 was applied to generating abstract models for delay ta-bisimulation. The work [4] describes its implementation for the case of strong time-abstracted bisimulation. A solution to the problem of computing differences of classes while generating a strong time-abstracted bisimulating model

[18] The actions of $\mathcal{A}$ can be taken as labels only, if a *discrete model* is to be built.

```
1.  Π := Π0; reachable := {initial_class}; stable := ∅;
2.  while (∃X ∈ reachable \ stable) do
3.      C_X := Split(X, Π);
4.      if (C_X = {X}) then
5.          stable := stable ∪ {X};
6.          reachable := reachable ∪ the_successors_of_X_in_Π;
7.      else
8.          Y_X := {Y ∈ Π | Y has been split };
9.          reachable := reachable \ Y_X ∪ {Y ∈ C_X | initial_state ∈ Y};
10.         stable := stable \ the_predecessors_of_elements_of_Y_X_in_Π;
11.         Π := (Π \ Y_X) ∪ C_X;
12.     end if;
13. end do;
```

Fig. 3. A generic minimization algorithm

was presented in [79]. Similarly to that of [87], it consists in starting from an initial partition of the state space respecting the invariants and enabling conditions of $\mathcal{A}$. An unstable class Y is refined simultaneously w.r.t. all its time-, or action a-successors (on a given action a). Since their inverse images form a partition of Y, differences do not need to be computed.

The algorithm of Fig. 3 was modified to build other kinds of abstract models, preserving more restricted classes of properties. The solutions differ in the definitions of the partition of Q, the stability condition and the function *Split* used to refine unstable classes. Below, we sketch the main of them.

Simulating models. The paper [32] shows a technique of building *simulating models*, preserving ACTL* properties. Comparing with bisimulating models, the condition on the successor relation between classes is relaxed, i.e., for each two reachable classes the condition U needs to be satisfied. The algorithm operates on a *cor*-partition $\Pi \subseteq 2^Q \times 2^Q$, consisting of pairs (Y, Y^{cor}) whose first elements form a partition of Q. Unstability of a class w.r.t. its successor can result in splitting one or both the classes, or modifying their *cor*s. The problem of computing differences for these models (for any ta-simulation) has not been solved so far.

Pseudo-bisimulating models. The paper [65] introduces *pseudo-bisimulating models*, preserving reachability properties. The main idea behind the definition consists in relaxing the condition on the transition relation on bisimulating models, formulated for all the predecessors of each abstract state, such that it applies only to one of them, reachable from the beginning state in the minimal number of steps. In this case, the algorithm deals with a *d-partition* $\Pi \subseteq 2^Q \times (\mathbb{N} \cup \{\infty\})$, which is a set of pairs $(Y, dpt(Y))$, whose first elements build a partition of Q. Unstability of a class w.r.t. its successor $(Y, dpt(Y))$ results in refining a predecessor of $(Y, dpt(Y))$ which is assumed to be reachable in the minimal number of steps. The models are usually built for the strong ta-bisimulation, using the method of avoiding computing differences given in [79].

Pseudo-simulating models. Both the solutions for simulating and pseudo-bisimulating models were combined, resulting in a definition of reachability-preserving *pseudo-simulating models* [66].

Other minimization techniques. Besides of the methods based on the minimization algorithms presented above, some other solutions exist. The paper [76] describes a partition refinement technique, operating on a product of the specifications of a system and a property, and exploiting splitting histories, whereas [44] presents a method of building reachability-preserving abstract models, exploiting (timed and untimed) histories of concrete states.

5.4 Forward-reachability graphs for TA

Verification based on reachability analysis is usually performed on an abstract model known as *simulation graph* or *forward-reachability graph*. The nodes of this graph can be defined as (not necessarily convex) sets of detailed regions [21] or as regions [30, 46, 88]. Usually, the latter approach is used, which follows from a convenient representation of zones by Difference Bound Matrices [34]. In this case, the simulation graph can be defined as the smallest graph $G = (W, w^0, \rightarrow)$ such that

- $w^0 = (l^0, Z^0)$ with $Z^0 = v^0{\nearrow} \cap [\![\mathcal{I}(l^0)]\!]$;
- for any $a \in A$ such that $e : l \stackrel{a,\mathfrak{cc},X}{\longrightarrow} l' \in \mathrm{E}$, and any $w = (l, Z) \in W$, if $Succ_a((l, Z)) \neq \emptyset$, then $w' = Succ_a((l, Z)) \in W$ and $w \stackrel{a}{\rightarrow} w'$, for $Succ_a((l, Z)) = (l', ((Z \cap [\![\mathfrak{cc}]\!])[X := 0]){\nearrow} \cap [\![\mathcal{I}(l')]\!])$.

The graph is usually generated using a forward-reachability algorithm which, starting from w^0, successively computes all the successors $Succ_a(w)$ for all $w \in W$ generated in earlier steps. The process can be terminated if a state satisfying a given property is reached before the whole graph is built. This is called *on-the-fly* reachability analysis. The simulation graph of [21] is finite, since the number of detailed regions is so. However, when generated in the above-presented manner, finiteness of the graph needs to be ensured. This is done by applying an *extrapolation abstraction* whose general idea is based on the fact that for any constraint of $\mathcal{A}$ involving a clock x, the exact value of $v(x)$ is insignificant if greater than the maximal value this clock is compared with (see [13, 30] for details). Other abstractions, enabling to reduce the size of the graph still enabling reachability checking, were presented in [30, 46, 88]. There are also many solutions aimed at reducing the memory usage while generating models [12, 30, 46].

5.5 On-the-fly Verification for TA

When the methods described above are used for verifying temporal properties, they require, in principle, to build a model first, and then to check a formula over this model. This can obviously make verification infeasible, especially when

the size of the model is prohibitive. Therefore, there are approaches which offer on-the-fly solutions, i.e., a formula is checked over a model while its construction. Bouajjanni et al [21] define an algorithm, which starts with building a simulation graph for a TA. Then, cycles of this graph are refined. This process is guided by a formula. If a stable cycle is found, then this means that the formula holds and at that point the verification ends.

Another solution has been suggested by Dickhofer and Wilke [33] and Henzinger et al. [45]. The idea follows the standard approach to automata-theoretic model checking for CTL. So, first an automaton accepting all the models for a TCTL formula is built and the product of this automaton with the automaton corresponding to the detailed region graph is constructed while its non-emptiness is checked [33]. The method of [45] is slightly different as the product is constructed without building the automaton for a formula first.

6 Symbolic Data Structures and Verification

To store and operate on abstract models usually Difference Bound Matrices [34] are used for state classes of TPN's [35] or regions of TA [34]. To our knowledge, there are very few approaches to BDD- or SAT-based verification of TPN's [19], which mainly consist in describing the state space of a net in a way which enables to use an existing symbolic tool. However, such approaches to verification of untimed Petri nets are known [26, 29, 52, 53, 60, 62, 77], but a discussion of them goes beyond the scope of this paper. Therefore, in what follows we consider symbolic data structures used for verification of TA. To this aim we list BDD-based methods and focus on the most recent SAT-related techniques for TA.

6.1 BDD- and SAT-Based Methods for TA - Overview

BDD-Based Methods. A standard symbolic approach to representation of state spaces and model checking of untimed systems is based on Binary Decision Diagrams (BDD's) [24]. A similar approach is to apply BDD's for encoding discretizations of TA using the so-called Numeric Decision Diagrams [9]. Discretizations of TA can be also implemented using propositional formulas (see Section 6.2).

Another approach follows the solution suggested by Henzinger et al. [40], where the characteristic function of a set of states is a formula in separation logic (SL)[19]. SL formulas can be represented using Difference Decision Diagrams (DDD's) [54, 55]. A DDD is a data structure using separation predicates with the ordering of predicates induced by the ordering of the clocks. A similar approach is taken in [73], where a translation from quantified SL to quantified boolean logic [78] is exploited.

One can use also Clock Difference Diagrams (CDD's) to symbolically represent unions of regions [11]. Each node of a CDD is labelled with the difference

[19] SL extends the propositional logic by clock constraints.

of clock variables, whereas the outgoing edges with intervals bounding this difference. Alternatively, model checkers are based on data structures called Clock Restriction Diagrams (CRD) [85]. CRD is like CDD except for the fact that for each node the outgoing edges are labelled with an upper bound rather than with the interval of the corresponding difference of clock variables.

Another recent symbolic approach has been motivated by a dramatic increase in efficiency of SAT-solvers, i.e., algorithms solving the satisfiability problem for propositional formulas [93].

SAT-Based Methods. The main idea of SAT-based methods consists in translating the model checking problem for a temporal logic to the problem of satisfiability of a formula in propositional or separation logic. This formula is typically obtained by combining an encoding of the model and of the temporal property. In principle, there are two different approaches. In the first one, a model checking problem for TCTL (or reachability properties) is translated to a formula in separation logic [10, 57, 75] or quantified separation logic [73] and then either solved by MathSAT[20] or translated further to propositional logic and solved by SAT-solver. The second approach exploits a translation of the model checking problem from TCTL to CTL and then further to a propositional formula [63].

On the other hand, the approaches to SAT-based symbolic verification can be viewed as unbounded (UMC) or bounded (BMC). UMC [73] is for unrestricted TCTL (or timed μ-calculus) on the whole model, whereas BMC [57, 63] applies to an existential fragment of TCTL (i.e., TECTL) on a part of the model. However, it is possible to use the bounded approach for verifying universal properties as well, which is shown for unreachability properties in [92]. In what follows, we focus on BMC for TECTL as well as for unreachability properties of TA.

6.2 BMC for Timed Automata

BMC consists in translating the model checking problem of an existential TCTL formula (i.e., containing only existential quantifiers) to the problem of satisfiability of a propositional formula. This translation is based on bounded semantics satisfaction, which, instead of using possibly infinite paths, is limited to finite prefixes only. Moreover, it is known that the translation of the existential path quantifier can be restricted to finitely many computations [62].

In this section we describe how to apply BMC to TECTL. The main idea of our method consists in translating the TECTL model checking problem to the model checking problem for a branching time logic [3, 79] and then in applying BMC for this logic [62].

We start with showing a discretization of TA and a translation from TCTL to slightly modified CTL (called $\mathrm{CTL}_{\mathrm{y}}$) on such discretized models. Then, we present BMC for $\mathrm{CTL}_{\mathrm{y}}$, and a BMC method of checking unreachability for TA.

[20] MathSAT is a solver checking satisfiability of SL.

Discretization of TA. We define a discretized model for a timed automaton, which is based on the discretization of [92]. The idea behind this method is to represent detailed zones of a timed automaton by one or more (but finitely many) specially chosen representatives.

Let $\mathcal{A} = (A, L, l^0, E, \mathcal{X}, \mathcal{I})$ be a timed automaton with $n_{\mathcal{X}}$ clocks, $V_{\mathcal{A}}$ be a valuation function, and φ be a TCTL formula. As before, let $M_c(\mathcal{A}) = (F_c(\mathcal{A}), V_c)$ be the concrete model for $\mathcal{A}$. We choose the discretization step $\Delta = 1/d$, where d is a fixed even number[21] greater than $2n_{\mathcal{X}}$. The *discretized clock space* is defined as $\mathbb{D}^{n_{\mathcal{X}}}$, where $\mathbb{D} = \{k\Delta \mid 0 \leq k \leq 2c_{max}(\varphi) + 2\}$. This means that the clocks cannot go beyond $2c_{max}(\varphi) + 2$, which follows from the fact that for evaluating the TCTL formula φ over diagonal-free timed automata we do not need to distinguish between clock valuations above $c_{max}(\varphi) + 1$. Similarly, the maximal values of time delays can be restricted to $c_{max}(\varphi) + 1$, since otherwise they would make the values of clocks greater than $c_{max}(\varphi) + 1$. Thus, the set of values that can change a valuation in a detailed zone, is defined as $\mathbb{E} = \{k\Delta \mid 0 \leq k\Delta < c_{max}(\varphi) + 1\}$[22]. To make sure that the above two definitions can be applied we will guarantee that before any time transition, the value of every clock does not exceed $c_{max}(\varphi) + 1$.

Next, we define the set $\mathbb{U}$ of valuations that are used to 'properly' represent detailed zones in the discretized model, i.e., we take a subset of the valuations v of $\mathbb{D}^{n_{\mathcal{X}}}$ that preserve time delays by insisting that either the values of all the clocks in v are only even or only odd multiplications of Δ. To preserve action successors we will later use 'adjust' transitions. The set $\mathbb{U}$ is defined as follows:

$$\mathbb{U} = \{u \in \mathbb{D}^{n_{\mathcal{X}}} \mid (\forall x \in \mathcal{X})(\exists k \in \mathbb{N})u(x) = 2k\Delta \vee (\forall x \in \mathcal{X})(\exists k \in \mathbb{N})u(x) = (2k+1)\Delta\}$$

Now, we are ready to define a *discretized model* for $\mathcal{A}$, that is later used for checking reachability and unreachability of a propositional formula p, which, as we already mentioned, is expressed by the formula CTL $\varphi = \mathrm{EF}p$ interpreted over runs with unrestricted time-successor steps. Note that in this case $c_{max}(\varphi)$ depends only on $\mathcal{A}$.

Definition 7 (Discretized Model). *The* discretized model *for $\mathcal{A}$ is a structure $DM(\mathcal{A}) = ((S, s^0, \rightarrow_d), V_d)$, where $S = L \times \mathbb{D}^{n_{\mathcal{X}}}$, $s^0 = (l^0, v^0)$ is the initial state, the labelled transition relation $\rightarrow_d \subseteq S \times (A \cup \mathbb{E} \cup \{\epsilon\}) \times S$ is defined as*

1. $(l, v) \xrightarrow{a}_d (l', v')$ *iff* $(l, v) \xrightarrow{a} (l', v')$ *in* $F_c(\mathcal{A})$, *for* $a \in A$, *(action transition)*
2. $(l, v) \xrightarrow{\delta}_d (l, v')$ *iff* $(\forall x \in \mathcal{X})(v(x) \leq c_{max}(\varphi) + 1)$, $v' = v + \delta$ *and* $v' \in [\![\mathcal{I}(l)]\!]$, *for* $\delta \in \mathbb{E}$ *(time delay transition),*
3. $(l, v) \xrightarrow{\epsilon}_d (l, v')$ *iff* $v' \in \mathbb{U}^{n_{\mathcal{X}}}$, $(\forall x \in \mathcal{X})(v'(x) \leq c_{max}(\varphi) + 1)$, *and* $v \simeq_{\mathcal{C}_{\mathcal{A},\varphi}} v'$ *(adjust transition).*

and the valuation function $V_d : S \to 2^{PV}$ is given by $V_d((l, v)) = V_{\mathcal{A}}(l)$.

[21] A good choice for d is the minimal such a number, which equals to 2^l for some l.
[22] By $\mathbb{E}_+$ we denote $\mathbb{E} \setminus \{0\}$.

Notice that the transitions in $DM(\mathcal{A})$ are labelled with actions of A, time delays of $\mathbb{E}$, or the epsilon label $\epsilon \notin A \cup \mathbb{E}$. The first two types of labels correspond exactly to labels used in the concrete model. The adjust transitions are used for moving within detailed zones to a valuation in $\mathbb{U}^{n,x}$. The reason for defining action and adjust transitions separately consists in increasing efficiency of the implementation for checking reachability in timed automata.

However, for verification of more complex TCTL formulas φ we need to put some restrictions on $DM(\mathcal{A})$ by defining the restricted discretized (r-discretized) model $DM_R(\mathcal{A}) = ((S', s^0, \rightarrow'_d), V'_d)$, where $S' = L \times \mathbb{U}^{n,x}$, $V'_d = V_d \cap S'$, and the transition relation $\rightarrow'_d \subseteq S' \times (A \cup \{\tau\}) \times S'$ is given as

1. $(l, v) \xrightarrow{\tau}{}'_d (l, v')$ iff $(l, v) \xrightarrow{\delta}_d (l, v')$ for some $\delta \in \mathbb{E}_+$, and
   if $(l, v) \xrightarrow{\delta'}_d (l, v'') \xrightarrow{\delta''}_d (l, v')$ with $\delta', \delta'' \in \mathbb{E}$ and $(l, v'') \in S$, then $v \simeq_{\mathcal{C}_{\mathcal{A},\varphi}} v''$ or $v' \simeq_{\mathcal{C}_{\mathcal{A},\varphi}} v''$, and
   if $v \simeq_{\mathcal{C}_{\mathcal{A},\varphi}} v'$, then $v \simeq_{\mathcal{C}_{\mathcal{A},\varphi}} v' + \delta''$ for each $\delta'' \in \mathbb{E}_+$ (time successor),
2. $(l, v) \xrightarrow{a}{}'_d (l', v')$ iff (l, v) is not boundary[23] and $((l, v) \xrightarrow{a}_d; \xrightarrow{\epsilon}_d (l', v')$ or $(l, v) \xrightarrow{\tau}{}'_d; \xrightarrow{a}_d; \xrightarrow{\epsilon}_d (l', v'))$, for $a \in A$ (action successor).

Intuitively, time successor corresponds to a move by time delay transition with the smallest δ to another region (if not final), whereas action successor corresponds to a move by action transition (adjusted by ϵ-transition to be in $\mathbb{U}^{n,x}$), taken from non-boundary regions, possibly preceded by the time successor step. Since our definition is based on the notion of the detailed region graph [3], it is easy to notice that $DM_R(\mathcal{A})$ is its discretization, and as such, it can be used for checking the TCTL formula φ.

Translation from TCTL to CTL. Rather than showing BMC directly for TECTL over timed automata, which is quite a complex task, we first discuss a translation from TCTL to a slightly modified CTL (CTL_y) and then BMC for ECTL_y. In general, the model checking problem for TCTL can be translated to the model checking problem for a fair version of CTL [3]. However, since we have assumed that we deal with progressive timed automata only, we can define a translation to the CTL_y model checking problem [79].

The idea is as follows. Given a timed automaton $\mathcal{A}$, a valuation function $V_\mathcal{A}$, and a TCTL formula φ. First, we extend $\mathcal{A}$ with a new clock, action, and transitions to obtain an automaton $\mathcal{A}_\varphi$. The aim of the new transitions is to reset the new clock, which corresponds to all the timing intervals appearing in φ. These transitions are used to start the runs over which subformulas of φ are checked. Then, we construct the r-discretized model for $\mathcal{A}_\varphi$ and augment its valuation function. Finally, we translate the TECTL formula φ to an ECTL_y formula $\psi = \text{cr}(\varphi)$ such that model checking of φ over the r-discretized model of $\mathcal{A}$ can be reduced to model checking of ψ over the r-discretized model of $\mathcal{A}_\varphi$.

Formally, let $\mathcal{X}$ be the set of clocks of $\mathcal{A}$, and $\{I_1, \ldots, I_r\}$ be a set of the non-trivial intervals appearing in φ. The automaton $\mathcal{A}_\varphi$ extends $\mathcal{A}$ such that

[23] A state (l, v) is boundary if for any $\delta \in \mathbb{E}_+$, $\neg(v \simeq_{\mathcal{C}_{\mathcal{A},\varphi}} v + \delta)$

– the set of clocks $\mathcal{X}' = \mathcal{X} \cup \{y\}$,
– the set of actions $A' = A \cup \{a_y\}$,
– the transition relation $E' \subseteq L \times A' \times \mathcal{C}_{\mathcal{X}'} \times 2^{\mathcal{X}'} \times L$ is defined as follows $E' = E \cup \{l \overset{a_y, true, \{y\}}{\longrightarrow} l \mid l \in L\}$.

Let $DM_R(\mathcal{A}_\varphi) = ((S, s^0, \rightarrow_d), V_d)$ be the r-discretized model for $\mathcal{A}_\varphi$. Denote by $\rightarrow_{\mathcal{A}}$ the part of $\rightarrow_d$, where transitions are labelled with elements of $A \cup \{\tau\}$, and by $\rightarrow_y$ the transitions that reset the clock y, i.e., labelled with a_y.

Next, we extend the set of propositional variables PV to PV' and the valuation function V_d to V. By $\wp_{y \in I_i}$ we denote a new proposition for every interval I_i appearing in φ, and by PV_φ the set of the new propositions. The proposition $\wp_{y \in I_i}$ is true at a state (l, v) of $DM_R(\mathcal{A}_\varphi)$ if $v(y) \in I_i$. Let V_φ be a function labelling each state of $DM_R(\mathcal{A}_\varphi)$ with the set of propositions from PV_φ true at that state and labelling each boundary state with $\wp_b$. Next, set $PV' = PV \cup PV_\varphi \cup \{\wp_b\}$ and define the valuation function $V : S \rightarrow 2^{PV'}$ as $V = V_d \cup V_\varphi$.

In order to translate a TCTL formula φ to the corresponding CTL formula ψ we need to modify the language of CTL to $\mathrm{CTL_y}$ by reinterpreting the next-time operator, denoted now by X_y. This language is interpreted over r-discretized models for $\mathcal{A}_\varphi$, defined above, where we assume that r is the number of non-trivial intervals appearing in φ. The modality X_y is interpreted only over the new transitions that reset the new clock y, whereas the other operators are interpreted over all the transitions except for the new ones. Formally, for $\wp \in PV'$, the set of $\mathrm{CTL_y}$ formulas is defined inductively as follows:

$$\alpha := \wp \mid \neg\wp \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \mathrm{X}_y\alpha \mid \mathrm{E}(\alpha \mathrm{U} \alpha) \mid \mathrm{E}(\alpha \mathrm{R} \alpha) \mid \mathrm{A}(\alpha \mathrm{R} \alpha) \mid \mathrm{A}(\alpha \mathrm{U} \alpha).$$

A *path* in $DM_R(\mathcal{A}_\varphi)$ is a maximal sequence $\pi = (s_0, s_1, \ldots)$ of states such that $s_i \rightarrow_{\mathcal{A}} s_{i+1}$ for each $i \in \mathbb{N}$. The relation $\models$ is defined like in Section 4.1 for all the $\mathrm{CTL_y}$ formulas, except for X_y, which is given as follows: $(l, v) \models \mathrm{X}_y\alpha$ iff $(l, v[\{y\} := 0)] \models \alpha$. Next, the TCTL formula φ is translated inductively to the $\mathrm{CTL_y}$ formula $\mathrm{cr}(\varphi)$ as follows:

– $\mathrm{cr}(\wp) = \wp$ for $\wp \in PV'$,
– $cr(\neg\wp) = \neg cr(\wp)$,
– $\mathrm{cr}(\alpha \vee \beta) = \mathrm{cr}(\alpha) \vee \mathrm{cr}(\beta)$,
– $\mathrm{cr}(\alpha \wedge \beta) = \mathrm{cr}(\alpha) \wedge \mathrm{cr}(\beta)$,
– $\mathrm{cr}(\mathrm{O}(\alpha \mathrm{U}_{I_i} \beta)) = \mathrm{X}_y\big(\mathrm{O}(\mathrm{cr}(\alpha) \mathrm{U} (\mathrm{cr}(\beta) \wedge \wp_{y \in I_i} \wedge (\wp_b \vee \mathrm{cr}(\alpha))))\big)$,
– $\mathrm{cr}(\mathrm{O}(\alpha \mathrm{R}_{I_i} \beta)) = \mathrm{X}_y\big(\mathrm{O}(\mathrm{cr}(\alpha) \mathrm{R} (\neg\wp_{y \in I_i} \vee (\mathrm{cr}(\beta) \wedge (\wp_b \vee \mathrm{cr}(\alpha))))))\big)$, for $\mathrm{O} \in \{\mathrm{E}, \mathrm{A}\}$.

It is easy to show that the validity of the TCTL formula φ over the concrete model of $\mathcal{A}$ is equivalent to the validity of the corresponding $\mathrm{CTL_y}$ formula $\mathrm{cr}(\varphi)$ over the r-discretized model of $\mathcal{A}_\varphi$ with the extended valuation function [3].

Next, we show a BMC method for $\mathrm{ECTL_y}$ over r-discretized models for TA. Since we have defined a translation from TCTL to $\mathrm{CTL_y}$, we obtain a BMC method for TECTL.

BMC **for ECTL$_{\mathbf{y}}$.** In this section we present a SAT-based approach to ECTL$_{\mathrm{y}}$ model checking over r-discretized models for timed automata, to which we refer as to models from now on. We start with giving a *bounded semantics* for ECTL$_{\mathrm{y}}$ in order to define the *bounded model checking problem* and to translate it subsequently into a satisfiability problem [62].

Let φ be a TECTL formula, $\psi = \mathrm{cr}(\varphi)$, and $M = ((S, s^0, \rightarrow_d), V)$ be a r-discretized model for $\mathcal{A}_\varphi$ with the extended valuation function.

We start with some auxiliary definitions. For $k \in \mathbb{N}_+$ a *k-path* in M is finite sequence of $k+1$ states $\pi = (s_0, s_1, \ldots, s_k)$ such that $(s_i, s_{i+1}) \in \rightarrow_{\mathcal{A}}$ for each $0 \leq i \leq k$. For a k-path $\pi = (s_0, s_1, \ldots, s_k)$, let $\pi(i) = s_i$ for each $i \leq k$. By $\Pi_k(s)$ we denote the set of all the k-paths starting at s. This is a convenient way of representing a k-bounded subtree rooted at s of the tree resulting from unwinding the model M from s.

Definition 8 (k-model). *A k-model for M is a structure $M_k = ((S, s^0, P_k), V)$, where P_k is the set of all the k-paths of M, i.e., $P_k = \bigcup_{s \in S} \Pi_k(s)$.*

Define a function $loop : P_k \rightarrow 2^{\mathbb{N}}$ as: $loop(\pi) = \{l \mid 0 \leq l \leq k \wedge \pi(k) \rightarrow_{\mathcal{A}} \pi(l)\}$. Satisfaction of the temporal operator R on a k-path π in the bounded case can depend on whether or not π represents a path[24], i.e., $loop(\pi) \neq \emptyset$,

The main reason for reformulating the semantics of the modalities in the following definition in terms of elements of k-paths rather than elements of S or Π is to restrict the semantics to a part of the model.

Definition 9 (k-bounded semantics for ECTL$_{\mathrm{y}}$). *Let M_k be a k-model and α, β be ECTL$_{\mathrm{y}}$ subformulas of ψ. $M_k, s \models \alpha$ denotes that α is true at the state s of M_k. M_k is omitted if it is clear from the context. The relation $\models$ is defined inductively as follows:*

$s \models \wp$ *iff* $\wp \in V(s)$
$s \models \neg\wp$ *iff* $\wp \notin V(s)$,
$s \models \alpha \wedge \beta$ *iff* $s \models \alpha$ *and* $s \models \beta$,
$s \models \alpha \vee \beta$ *iff* $s \models \alpha$ *or* $s \models \beta$,
$s \models \mathrm{X}_y\alpha$ *iff* $\exists s' \in S \left(s \rightarrow_y s' \text{ and } s' \models \alpha\right)$,
$s \models \mathrm{E}(\alpha \mathrm{U} \beta)$ *iff* $\exists \pi \in \Pi_k(s)\Big(\exists_{0 \leq j \leq k}\Big(\pi(j) \models \beta \text{ and } \forall_{0 \leq i < j}\, \pi(i) \models \alpha\Big)\Big)$,
$s \models \mathrm{E}(\alpha \mathrm{R} \beta)$ *iff* $\exists \pi \in \Pi_k(s)\Big(\Big(\exists_{0 \leq j \leq k}\Big(\pi(j) \models \alpha \text{ and } \forall_{0 \leq i < j}\, \pi(i) \models \beta\Big)\Big)$ *or*
$\Big(\forall_{0 \leq j \leq k}\, \pi(j) \models \beta \text{ and } loop(\pi) \neq \emptyset\Big)\Big)$.

Next, we describe how the model checking problem ($M \models \psi$) can be reduced to the bounded model checking problem ($M_k \models \psi$). In this setting we can prove that in some circumstances satisfiability in the $|M|$-bounded semantics is equivalent to the unbounded one.

Theorem 1. *Let $M = ((S, s^0, \rightarrow_d), V)$ be a model, ψ be an ECTL$_{\mathrm{y}}$ formula and $k = |M|$. Then, $M, s^0 \models \psi$ iff $M_k, s^0 \models \psi$.*

[24] Note that a path is infinite by definition.

The rationale behind the method is that for particular examples checking satisfiability of a formula can be done on a small fragment of the model.

Next, we show how to translate the model checking problem for $\mathrm{ECTL_y}$ on a k-model to a problem of satisfiability of some propositional formula. Our method is based on [62], but we use the operator ER rather than less expressive EG. Proofs of correctness of our approach are extensions of the corresponding proofs in [62]. We assume the following definition of a submodel.

Definition 10. *Let $M_k = ((S, s^0, P_k), V)$ be a k-model of M. A structure $M'_k = ((S', s^0, P'_k), V')$ is a* submodel *of M_k if $P'_k \subseteq P_k$, $S' = States(P'_k)$, and $V' = V|_{S'}$, where $States(P'_k) = \{s \in S \mid (\exists \pi \in P'_k)(\exists i \leq k)\ \pi(i) = s\}$.*

The bounded semantics of $\mathrm{ECTL_y}$ over submodels M'_k is defined as for M_k (see Def. 9). Our present aim is to give a bound for the number of k-paths in M'_k such that the validity of ψ in M_k is equivalent to the validity of ψ in M'_k. Let $\mathrm{F}_{\mathrm{ECTL}y}$ be a set of the formulas of $\mathrm{ECTL_y}$.

Definition 11. *Define a function $f_k : \mathrm{F}_{\mathrm{ECTL}y} \to \mathbb{N}$ as follows:*

- $f_k(\wp) = f_k(\neg\wp) = 0$, *where* $\wp \in PV$,
- $f_k(\alpha \vee \beta) = max\{f_k(\alpha), f_k(\beta)\}$,
- $f_k(\alpha \wedge \beta) = f_k(\alpha) + f_k(\beta)$,
- $f_k(\mathrm{X}_y\alpha) = f_k(\alpha)$,
- $f_k(\mathrm{E}(\alpha \mathrm{U} \beta)) = k \cdot f_k(\alpha) + f_k(\beta) + 1$,
- $f_k(\mathrm{E}(\alpha \mathrm{R} \beta)) = k \cdot f_k(\beta) + f_k(\alpha) + 1$.

The function f_k determines the number of k-paths of a submodel M'_k sufficient for checking an $\mathrm{ECTL_y}$ formula.

The main idea is that we can check ψ over M_k by checking the satisfiability of a propositional formula $[M, \psi]_k = [M^{\psi, s^0}]_k \wedge [\psi]_{M_k}$, where the first conjunct represents (part of) the model under consideration and the second a number of constraints that must be satisfied on M_k for ψ to be satisfied.

Once this translation is defined, checking satisfiability of an $\mathrm{ECTL_y}$ formula can be done by means of a SAT-checker. Although from a theoretical point of view the complexity of this operation is no easier, in practice the efficiency of modern SAT-checkers makes the process worthwhile in many instances.

We now give details of this translation. We begin with the encoding of the transitions in the model under consideration. Since the set of states S of our model is finite, every element of S can be encoded as a bit vector of a length depending on the number of locations in L, the size of the set $\mathbb{D}$ and $c_{max}(\varphi)$. We do not give details of this encoding here. The interested reader is referred to [63, 92]. Each state s can be represented by a vector $w = (w[1], \ldots, w[l])$ (called a *global state variable*), where each $w[i]$ for $i = 1, \ldots, l$ is a propositional variable (called state variable). Notice that we distinguish between states s encoded as sequences of 0's and 1's and their representations in terms of propositional variables $w[i]$. A finite sequence $(w_0, \ldots, w_k)$ of global state variables is called a *symbolic k-path*.

In general we shall need to consider not just one but a number of symbolic k-paths. This number depends on the formula ψ under investigation, and it is returned as the value $f_k(\psi)$ of the function f_k.

To construct $[M, \psi]_k$, we first define a propositional formula $[M^{\psi,s^0}]_k$ that constrains the $f_k(\psi)$ symbolic k-paths to be valid k-paths of M_k. The j-th symbolic k-path is denoted as $w_{0,j}, \ldots, w_{k,j}$, where $w_{i,j}$ are global state variables for $1 \leq j \leq f_k(\psi)$, $0 \leq i \leq k$. Let PV_s be a set of state variables, $\mathcal{FORM}$ be a set of propositional formulas over PV_s, and let $lit : \{0,1\} \times PV_s \to \mathcal{FORM}$ be a function defined as follows: $lit(0, \wp) = \neg\wp$ and $lit(1, \wp) = \wp$. Furthermore, let w, v be global state variables. We define the following propositional formulas:

- $I_s(w) := \bigwedge_{i=1}^{l} lit(s[i], w[i])$ (encodes the state s of the model M, i.e., $s[i] = 1$ is encoded by $w[i]$, and $s[i] = 0$ is encoded by $\neg w[i]$).
- $\wp(w)$ is a formula over w, which is true for a valuation s_w of w iff $\wp \in V(s_w)$, where $\wp \in PV'$ (see page 26),
- $H(w, v) := \bigwedge_{i=1}^{l} w[i] \Leftrightarrow v[i]$ (equality of the two state encodings),
- $\mathcal{R}(w, v)$ is a formula over w, v, which is true for two valuations s_w of w and s_v of v iff $s_w \to_{\mathcal{A}} s_v$ (encodes the transition relation of the paths),
- $R_y(w, v)$ is a formula over w, v, which is true for two valuations s_w of w and s_v of v iff $s_w \to_y s_v$ (encodes the transitions resetting the clock y),
- $L_{k,j}(l) := \mathcal{R}(w_{k,j}, w_{l,j})$, (encodes a backward loop from the k-th state to the l-th state in the symbolic k-path j, for $0 \leq l \leq k$).

The translation of $[M^{\psi,s^0}]_k$, representing the transitions in the k-model is given by the following definition.

Definition 12 (Encoding of Transition Relation). *Let $M_k = ((S, s^0, P_k), V)$ be the k-model of M, and ψ be an* $\mathrm{ECTL}_{\mathrm{y}}$ *formula. The propositional formula $[M^{\psi,s^0}]_k$ is defined as follows:*

$$[M^{\psi,s^0}]_k := I_{s^0}(w_{0,0}) \wedge \bigwedge_{j=1}^{f_k(\psi)} \bigwedge_{i=0}^{k-1} \mathcal{R}(w_{i,j}, w_{i+1,j})$$

where $w_{0,0}$, and $w_{i,j}$ for $0 \leq i \leq k$ and $1 \leq j \leq f_k(\psi)$ are global state variables. $[M^{\psi,s^0}]_k$ constrains the $f_k(\psi)$ symbolic k-paths to be valid k-paths in M_k.

The next step of our algorithm is to translate an $\mathrm{ECTL}_{\mathrm{y}}$ formula ψ into a propositional formula. We use $[\alpha]_k^{[m,n]}$ to denote the translation of an $\mathrm{ECTL}_{\mathrm{y}}$ subformula α of ψ at $w_{m,n}$ to a propositional formula, where $w_{m,n}$ are global state variables with $0 \leq m \leq k$ for $1 \leq n \leq f_k(\psi)$ (which correspond to $f_k(\psi)$ symbolic paths), and with $m = 0$ for $f_k(\psi)+1 \leq n \leq f_k(\psi)+r$ (which correspond to r global state variables[25] for representing states, where the clock y is reset). Note that the index n denotes the number of a symbolic path, whereas the index m the position at that path.

[25] Recall that r is the number of non-trivial intervals in φ, where $\psi = \mathrm{cr}(\varphi)$.

$$
\begin{aligned}
[\wp]_k^{[m,n]} &:= \wp(w_{m,n}),\\
[\neg\wp]_k^{[m,n]} &:= \neg\wp(w_{m,n}),\\
[\alpha \wedge \beta]_k^{[m,n]} &:= [\alpha]_k^{[m,n]} \wedge\ [\beta]_k^{[m,n]},\\
[\alpha \vee \beta]_k^{[m,n]} &:= [\alpha]_k^{[m,n]} \vee\ [\beta]_k^{[m,n]},\\
[\mathrm{X}_y\alpha]_k^{[m,n]} &:= \bigvee_{1\le j\le r} \Big(R_y(w_{m,n}, w_{0,f_k(\psi)+j}) \ \wedge\ [\alpha]_k^{[0,f_k(\psi)+j]} \Big),\\
[\mathrm{E}(\alpha \mathrm{U} \beta)]_k^{[m,n]} &:= \bigvee_{1\le i\le f_k(\psi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge\ \bigvee_{j=0}^{k} \Big([\beta]_k^{[j,i]} \ \wedge\ \bigwedge_{l=0}^{j-1} [\alpha]_k^{[l,i]} \Big) \Big),\\
[\mathrm{E}(\alpha \mathrm{R} \beta)]_k^{[m,n]} &:= \bigvee_{1\le i\le f_k(\psi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge\ \Big(\bigvee_{j=0}^{k} \Big([\alpha]_k^{[j,i]} \wedge\ \bigwedge_{l=0}^{j-1} [\beta]_k^{[l,i]} \Big) \ \vee\\
&\qquad\qquad \bigwedge_{j=0}^{k}\ [\beta]_k^{[j,i]} \ \wedge\ \bigvee_{l=0}^{k} L_{k,i}(l) \Big) \Big).
\end{aligned}
$$

Given the translations above, we can now check ψ over M_k by checking satisfiability of the propositional formula $[M^{\psi,s^0}]_k \wedge [\psi]_k^{[0,0]}$. The translation presented above can be shown to be correct and complete.

Theorem 2. *Let M be a model, M_k be a k-model of M, and ψ be an* ECTL_y *formula. Then, $M \models_k \psi$ iff $[\psi]_{M_k} \wedge [M^{\psi,s^0}]_k$ is satisfiable.*

We have all ingredients in place to give the algorithm for BMC of TECTL.

Definition 13. BMC *algorithm for* TECTL*:*

1. *Let φ be a* TECTL *formula and $\mathcal{A}$ be a timed automaton.*
2. *Let $M = ((S, s^0, \to_d), V)$ be the r-discretized model with the extended valuation function for $\mathcal{A}_\varphi$.*
3. *Let $\psi = \mathrm{cr}(\varphi)$ be the* ECTL_y *formula.*
4. *Set $k := 1$.*
5. *Select the k-model M_k.*
6. *Select the submodels M'_k of M_k with $|P'_k| \le f_k(\psi)$.*
7. *Encode the transition relation of all M'_k by a propositional formula $[M^{\psi,s^0}]_k$.*
8. *Translate ψ over all M'_k into a propositional formula $[\psi]_{M_k}$.*
9. *Check the satisfiability of $[M,\psi]_k := [M^{\psi,s^0}]_k \wedge [\psi]_{M_k}$.*
10. *If $[M,\psi]_k$ is satisfiable, then return $M_c(\mathcal{A}) \models \varphi$.*
11. *Set $k := k + 1$.*
12. *If $k = |M| + 1$, then return $M_c(\mathcal{A}) \not\models \varphi$ else go to 5.*

Checking Reachability with BMC**.** Reachability of a propositional formula p in a timed automaton $\mathcal{A}$ can be specified by the ECTL formula EFp. So, in principle, reachability can be verified using the above approach over $DM_R(\mathcal{A})$ slightly modified to incorporate zero time successor steps. It turns out, however, that a slight change in the technique can dramatically influence efficiency of the method in this case.

First of all, we consider k-paths over $DM(\mathcal{A})$, rather than over $DM_R(\mathcal{A})$, which means that action and adjust transitions are not combined, and the delay transition relation is transitive. Secondly, we use the notion of a *special k-path*, which satisfies the following conditions.

– It begins with the initial state.
– The first transition is a time delay one.
– Each time delay transition is directly followed by an action one.
– Each action transition is directly followed by an adjust one.
– Each adjust transition is directly followed by a time delay one.
– The above three rules do not apply only to the last transition of a special k-path.

Obviously, it is sufficient to use only one symbolic path to encode all the special k-paths. Thus, the reachability problem is translated to conjunction of the encoding of the symbolic path and the encoding of the propositional property p at the last state of that path. If this conjunction is satisfiable, then p is reachable. In Section 8 we discuss experimental results obtained using this method.

Checking Unreachability with BMC**.** Unreachability of a propositional formula p in a timed automaton $\mathcal{A}$ means that the ACTL formula AG$\neg p$ holds in $\mathcal{A}$. Again, for verification, we could check the ECTL formula EFp over a slightly modified $DM_R(\mathcal{A})$, but, in this case, we have to prove that this formula does not hold in the model. This is, obviously, one of the major problems with BMC, as in the worst case the algorithm needs to reach the upper bound for k, i.e., the size of the model.

There is, however, another approach to checking unreachability, which in many cases (the method is not complete) gives striking results. The idea is to use a SAT-solver to find a minimal (possible) k such that if $\neg p$ holds at all the k-paths, then it means that p is unreachable. The method described below finds such a k, if each path at which p holds only at the final state is finite. To this aim, a *free special k-path* is defined. It satisfies all the conditions on a special k-path except for the first one, i.e., it does not need to start at the initial state.

In addition we require that for a special k-path the following conditions hold:

– p holds only at the last state if the last transition is an action one,
– p holds only at the last two states if the last transition is an adjust one,
– p holds only at the last three states if the last transition is a time delay one.

Notice that if p holds in our model, then it holds at a path of length restricted by the length of a longest special k-path.

So, using one symbolic k-path, we encode all the free special k-paths in order to find the length of a longest one satisfying the above three conditions. If the above encoding is unsatisfiable for some $k = k_0$, then it means that we have found a longest free path. It is known that we can look for such a k by running the algorithm for the values of k satisfying $k \ mod \ 3 \ = \ 2$ only. Then, when we find k_0 for which the encoding is unsatisfiable, we can run the check for reachability of p up to $k = k_0 - 3$. If the reachability algorithm does not find the formula satisfiable for such a k, then it means that p is indeed unreachable.

Unfortunately, the above method is not complete, it fails when there are loops in the unreachable part of the state space involving states satisfying $\neg p$, from which a state satisfying p is reachable.

A solution to make the method complete by encoding that a free special path is loop-free i.e., no state repeats at the path, turns out to be ineffective in practice [92]. In Section 8 we discuss experimental results obtained using this method.

7 Existing Tools

Tools for TPN's. Some of the existing tools for Petri nets with time are listed below:

- **Tina** [16] - a toolbox for analysis of (time) Petri nets. It constructs state class graphs [14, 15] and performs LTL or reachability verification. In addition, Tina builds atomic state class graphs [17] to be used for verification of CTL formulas.
- **Romeo** [70] - a tool for time Petri nets analysis, which provides several methods for translating TPN's to TA [25, 49] and computation of state class graphs [35].
- **INA** (Integrated Net Analyser) - a Petri net analysis tool, supporting place/transition nets and coloured Petri nets with time and priorities. Among others, INA provides verification by analysis of paths for TPN's [67].
- **CPN Tools** [69] (a replacement for **Design/CPN**) - a software package for modelling and analysis of both timed and untimed Coloured Petri Nets, enabling their simulation, generating occurrence (reachability) graph, and analysis by place invariants.

Tools for TA. There are many tools using the approaches considered in this paper. Below, we list some of them and give pointers to the literature, where more detailed descriptions can be found.

- **Cospan** - a tool for verifying the behaviour of designs written in the industry standard design languages VHDL and Verilog. It implements an automata-based approach to model checking including an on-the-fly enumerative search (using zones in the timed case), as well as symbolic search using BDDs. A detailed description of timed verification can be found in [8].
- **Kronos** [91] is a tool which performs verification of TCTL using forward or backward analysis, and behavioural analysis, which consists in building the smallest finite quotient of a timed model (using minimization), and then checking whether the minimal model of the system simulates that of the specification. DBM's are used for representing zones.
- **UppAal2k** (a successor of UppAal) - a tool for modelling, simulation and verification of timed systems, appropriate for systems which can be described by a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables. Forward reachability analysis, deadlock detection and verification of properties expressible in a subset of TCTL are available. Many optimizations are implemented, e.g. application of Clock Difference Diagrams (CDD's) to represent unions of regions [11].

– **Red** is a fully symbolic model checker based on data structures called Clock Restriction Diagrams (CRD) [85]. It supports TCTL model checking and backward reachability analysis.
– **HyTech** [39] - an automatic tool for the analysis of embedded systems. Real-time requirements are specified in the logic TCTL and its modification - ICTL (*Integrator Computation Tree Logic*), used to specify safety, liveness, time-bounded and duration requirements of hybrid automata. Verification is performed by a successive approximation of the set of states satisfying the formula to be checked, by iterating boolean operations and weakest-precondition operations on regions (see [7]).
– **Rabbit** [18] - a tool for BDD-based verification of real-time systems, developed for an extension of TA, called Cottbus Timed Automata, and providing reachability analysis.
– **Verics** [31] - implements partition refinement algorithms and SAT-based BMC for verifying TCTL and reachability for timed automata and Estelle programs.

8 Experimental Results for Verifying TPN's and TA

In this section we compare experimental results for the four TPN's of Figure 1, and for the TA of Figure 2 modelling Fischer's mutual exclusion algorithm.

In the first table (Fig. 4) we give the sizes of several abstract models for all the nets, obtained using either state classes approaches (Yoneda's implementation, Tina), or minimization algorithms (Verics, Kronos) applied to translations[26] to timed automata as well as directly to the TA of Figure 2. The experiments were performed on a PC (Intel Pentium III 640MHz), with the assumed limit on the execution time (30 min) and memory required (128 MB RAM + 128 MB of swap space under Linux). For the Fischer's protocol, we give the sizes of models for 3 processes as well as for the maximal number of processes a model could be generated for. The abbreviation *nsp* means that the tool does not support verification of time Petri nets, where Lft of a transition is equal to ∞.

The models generated can be divided into three groups, the first of which contains the structures to be used for reachability checking: (strong) state class graphs (SCG and SSCG, resp.), geometric region graphs, forward-reachability graphs obtained using *inclusion* [30] and extrapolation abstractions (*forw -ax - ai*), and pseudo-simulating models generated for the semantics which collects together time- and transition steps (*ps- discrete*). The next group includes various kinds of bisimulating models built for the above semantics: (strong) atomic state class graphs (SASCG and *atomic*, resp.), and models denoted by *bis. discrete*. In a separate class are bisimulating models for the dense semantics considered in this paper (i.e., *bis. dense*).

Notice that the sizes of models for the nets 5a, 5b, and 5c are comparable for different approaches and there is no tool (approach), which would outperform the other ones w.r.t. the sizes of all the types of models. For MUTEX of 3

[26] Processes-as-clocks translations of [64] are used.

processes, all the algorithms give nearly the same results, but only Tina can generate models for MUTEX of 9 processes within the assumed time limit.

		Net 5a		Net 5b		Net 5c		Mutex Δ=1, δ=2			Mutex Δ=2, δ=1		
		states	edges	states	edges	states	edges	noP	states	edges	noP	states	edges
OBTAINED BY TPN - SPECIFIC METHODS													
Tina	SASCG	36	61	62	163	80	204	3	65	96	3	152	240
								9	81035	280170	7	73600	200704
Tina	SCG	18	26	34	58	50	76	3	65	96	3	152	240
								9	81035	280170	7	73600	200704
Tina	SSCG	21	29	39	63	60	93	3	65	96	3	152	240
								9	81035	280170	7	73600	200704
impl.[89]	atomic	53	95	64	179	168	363	3	nsp	nsp	3	nsp	nsp
impl.[89]	geometric	16	25	32	57	105	170	3	nsp	nsp	3	nsp	nsp
OBTAINED BY TPN $\rightarrow$TA TRANSLATIONS													
VerICS	bis. dense	54	80	135	230	186	323	3	77	108	3	200	312
								3	77	108	3	200	312
VerICS	bis. discr.	26	47	46	135	80	204	3	65	96	3	152	240
								3	65	96	3	152	240
VerICS	ps- discr.	21	34	13	22	53	121	3	65	96	3	152	204
								3	65	96	3	152	204
Kronos	bis. dense	51	77	134	229	185	321	3	77	108	3	200	312
								5	807	1590	4	1008	1856
Kronos	forw -ai -ax	37	42	37	42	26	40	3	214	321	3	613	1084
								5	33451	62223	4	12850	27848

Fig. 4. Experimental results for the nets in Fig. 1

In the second table (Fig. 5) we display the results of applying the BMC algorithm (for checking reachability and unreachability) to timed systems modelling Fischer's mutual exclusion. We verify that either mutual exclusion is violated for $\Delta = 2$ and $\delta = 1$, or is preserved for $\Delta = 1$ and $\delta = 2$. BMC is applied either directly to the TA, or to the timed automaton resulting from the translation of the TPN. For the case of $\Delta = 2$ and $\delta = 1$ we provide the time and memory resources needed to confirm satisfiability of the property on a special path of length $k = 17$, whereas for $\Delta = 1$ and $\delta = 2$ the time given is sum of the times required to check that a special free path of length $k = 44$ is the longest possible one and then to check satisfiability on the special path of that length. These experiments were performed on a PC (AMD Athlon XP 1800 - 1544MHz).

Notice that the BMC could verify MUTEX modelled by the TPN or the TA for $\Delta = 1, \delta = 2$ of 8 processes, and respectively of 104 and 310 processes, where the mutual exclusion was violated ($\Delta = 2$ and $\delta = 1$).

The above results show that verifying time Petri nets via translations to timed automata can sometimes give better results than using specific methods for nets. This is especially the case, when BMC is used for verification. Therefore, it seems interesting to investigate different combinations of specific methods for both timed automata and time Petri nets like for example BMC applied to models based on state class graphs rather that on r-discretized region graphs.

		TPN→TA				TA			
Parameters	**NoP**	variables	**clauses**	**sec**	**MB**	variables	**clauses**	**sec**	**MB**
$\Delta = 1$, $\delta = 2$	8	61530	176319	10890.1	61.31	36461	103228	2326.3	34.5
$\Delta = 2$, $\delta = 1$	8	22552	64442	8.0	21.9	13357	37666	0.7	20.5
$\Delta = 2$, $\delta = 1$	10	29918	86002	14.5	23.5	17283	49034	1.1	20.2
$\Delta = 2$, $\delta = 1$	50	378203	1118763	99.7	100.5	156941	459722	21.8	31.7
$\Delta = 2$, $\delta = 1$	104	1411156	4200809	1397.6	577.9	528136	1562194	218.8	75.9
$\Delta = 2$, $\delta = 1$	310	-	-	-	-	3873940	11557290	21723.6	648.3

Fig. 5. BMC of VERICS for Fischer's protocol modelled by TPN and TA

Acknowledgements: Many thanks to B. Berthomieu, O. H. Roux, M. Szreter, and B. Woźna for their comments that greatly helped to improve this paper. The authors are especially grateful to A. Zbrzezny for providing experimental results of BMC.

References

1. P. A. Abdulla and A. Nylén. Timed Petri Nets and BQOs. In *Proc. of ICATPN'01*, volume 2075 of *LNCS*, pages 53–70. Springer-Verlag, 2001.
2. R. Alur, C. Courcoubetis, and D. Dill. Model checking for real-time systems. In *Proc. of LICS'90*, pages 414–425. IEEE, 1990.
3. R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
4. R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. of RTSS'92*, pages 157–166. IEEE Comp. Soc. Press, 1992.
5. R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. Minimization of timed transition systems. In *Proc. of CONCUR'92*, volume 630 of *LNCS*, pages 340–354. Springer-Verlag, 1992.
6. R. Alur and D. Dill. Automata for modelling real-time systems. In *Proc. of ICALP'90*, volume 443 of *LNCS*, pages 322–335. Springer-Verlag, 1990.
7. R. Alur, T. Henzinger, and P. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Eng.*, 22(3):181–201, 1996.
8. R. Alur and R. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III*, volume 1066 of *LNCS*, pages 220–231. Springer-Verlag, 1996.
9. E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of Timed Automata. In *Proc. of HART'97*, volume 1201 of *LNCS*, pages 346–360. Springer-Verlag, 1997.
10. G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded model checking for timed systems. In *Proc. of FORTE'02*, volume 2529 of *LNCS*, pages 243–259. Springer-Verlag, 2002.
11. G. Behrmann, K. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using Clock Difference Diagrams. In *Proc. of CAV'99*, volume 1633 of *LNCS*, pages 341–353. Springer-Verlag, 1999.
12. J. Bengtsson. *Clocks, DBMs and States in Timed Systems*. PhD thesis, Dept. of Information Technology, Uppsala University, 2002.
13. J. Bengtsson and W. Yi. On clock difference constraints and termination in reachability analysis in Timed Automata. In *Proc. of ICFEM'03*, volume 2885 of *LNCS*, pages 491–503. Springer-Verlag, 2003.

14. B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using Time Petri Nets. *IEEE Trans. on Software Eng.*, 17(3):259–273, 1991.
15. B. Berthomieu and M. Menasche. An enumerative approach for analyzing Time Petri Nets. In *Proc. of the IFIP 9th World Computer Congress*, volume 9 of *Information Processing*, pages 41–46. North Holland/ IFIP, September 1983.
16. B. Berthomieu, P-O. Ribet, and F. Vernadat. The tool TINA - construction of abstract state spaces for Petri nets and Time Petri Nets. *International Journal of Production Research*, 2004. to appear.
17. B. Berthomieu and F. Vernadat. State class constructions for branching analysis of Time Petri Nets. In *Proc. of TACAS'03*, volume 2619 of *LNCS*, pages 442–457. Springer-Verlag, 2003.
18. D. Beyer. Rabbit: Verification of real-time systems. In *Proc. of the Workshop on Real-Time Tools (RT-TOOLS'01)*, pages 13–21, 2001.
19. A. Bobbio and A. Horváth. Model checking time Petri nets using NuSMV. In *Proc. of the 5th Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS5)*, pages 100–104, September 2001.
20. A. Bouajjani, J-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18:247–269, 1992.
21. A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model checking for real-time systems. In *Proc. of RTSS'97*, pages 232–243. IEEE Comp. Soc. Press, 1997.
22. H. Boucheneb and G. Berthelot. Towards a simplified building of Time Petri Nets reachability graph. In *Proc. of the 5th Int. Workshop on Petri Nets and Performance Models*, pages 46–55, October 1993.
23. F. Bowden. Modelling time in Petri nets. In *Proc. of the 2nd Australia-Japan Workshop on Stochastic Models (STOMOD'96)*, July 1996.
24. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8):677–691, 1986.
25. F. Cassez and O. H. Roux. Traduction structurelle des Réseaux de Petri Temporels vers le Automates Temporisés. In *Proc. of 4ieme Colloque Francophone sur la Modélisation des Systémes Réactifs (MSR'03)*. Hermes Science, October 2003.
26. G. Ciardo, G. Lüttgen, and R. Simniceanu. Efficient symbolic state-space construction for asynchronous systems. In *Proc. of ICATPN'00*, volume 1825 of *LNCS*, pages 103–122. Springer-Verlag, 2000.
27. J. Coolahan and N. Roussopoulos. Timing requirements for time-driven systems using augmented Petri nets. *IEEE Trans. on Software Eng.*, SE-9(5):603–616, 1983.
28. L. A. Cortés, P. Eles, and Z. Peng. Verification of real-time embedded systems using Petri net models and Timed Automata. In *Proc. of the 8th Int. Conf. on Real-Time Computing Systems and Applications (RTCSA'02)*, pages 191–199, March 2002.
29. J-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Pointrenaud, and P-A. Wacrenier. Data Decision Diagrams for Petri net analysis. In *Proc. of ICATPN'02*, volume 2360 of *LNCS*, pages 101–120. Springer-Verlag, 2002.
30. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Proc. of TACAS'98*, volume 1384 of *LNCS*, pages 313–329. Springer-Verlag, 1998.
31. P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS: A tool for verifying Timed Automata and Estelle specifications. In *Proc. of TACAS'03*, volume 2619 of *LNCS*, pages 278–283. Springer-Verlag, 2003.

32. P. Dembiński, W. Penczek, and A. Półrola. Verification of Timed Automata based on similarity. *Fundamenta Informaticae*, 51(1-2):59–89, 2002.
33. M. Dickhofer and T. Wilke. Timed Alternating Tree Automata: The automata-theoretic solution to the TCTL model checking problem. In *Proc. of ICALP'98*, volume 1664 of *LNCS*, pages 281–290. Springer-Verlag, 1998.
34. D. Dill. Timing assumptions and verification of finite state concurrent systems. In *Automatic Verification Methods for Finite-State Systems*, volume 407 of *LNCS*, pages 197 – 212. Springer-Verlag, 1989.
35. G. Gardey, O. H. Roux, and O. F. Roux. Using zone graph method for computing the state space of a Time Petri Net. In *Proc. of FORMATS'03*, volume 2791 of *LNCS*. Springer-Verlag, 2004.
36. Z. Gu and K. Shin. Analysis of event-driven real-time systems with Time Petri Nets. In *Proc. of DIPES'02*, volume 219 of *IFIP Conference Proceedings*, pages 31–40. Kluwer, 2002.
37. S. Haar, L. Kaiser, F. Simonot-Lion, and J. Toussaint. On equivalence between Timed State Machines and Time Petri Nets. Technical Report RR-4049, INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 Montbonnot-St-Martin, November 2000.
38. H-M. Hanisch. Analysis of place/transition nets with timed arcs and its application to batch process control. In *Proc. of ICATPN'93*, volume 691 of *LNCS*, pages 282–299. Springer-Verlag, 1993.
39. T. Henzinger and P. Ho. HyTech: The Cornell hybrid technology tool. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 265–293. Springer-Verlag, 1995.
40. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–224, 1994.
41. M. Huhn, P. Niebert, and F. Wallner. Verification based on local states. In *Proc. of TACAS'98*, volume 1384 of *LNCS*, pages 36–51. Springer-Verlag, 1998.
42. H. Hulgaard and S. M. Burns. Efficient timing analysis of a class of Petri Nets. In *Proc. of CAV'95*, volume 939 of *LNCS*, pages 923–936. Springer-Verlag, 1995.
43. R. Janicki. Nets, sequential components and concurrency relations. *Theoretical Computer Science*, 29:87–121, 1984.
44. I. Kang and I. Lee. An efficient state space generation for the analysis of real-time systems. In *Proc. of Int. Symposium on Software Testing and Analysis*, 1996.
45. O. Kupferman, T. A. Henzinger, and M. Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In *Proc. of CONCUR'96*, volume 1119 of *LNCS*, pages 514–529. Springer-Verlag, 1996.
46. K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *Proc. of RTSS'97*, pages 14–24. IEEE Comp. Soc. Press, 1997.
47. D. Lee and M. Yannakakis. On-line minimization of transition systems. In *Proc. of the 24th ACM Symp. on the Theory of Computing*, pages 264–274, May 1992.
48. J. Lilius. Efficient state space search for Time Petri Nets. In *Proc. of MFCS Workshop on Concurrency, Brno'98*, volume 18 of *ENTCS*. Elsevier Science Publishers, 1999.
49. D. Lime and O. H. Roux. State class timed automaton of a time Petri net. In *Proc. of the 10th Int. Workshop on Petri Nets and Performance Models (PNPM'03)*. IEEE Comp. Soc. Press, September 2003.
50. K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proc. of CAV'02*, volume 2404 of *LNCS*, pages 250–264. Springer-Verlag, 2002.

51. P. Merlin and D. J. Farber. Recoverability of communication protocols – implication of a theoretical study. *IEEE Trans. on Communications*, 24(9):1036–1043, 1976.
52. A. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In *Proc. of ICATPN'99*, volume 1639 of *LNCS*, pages 6–25. Springer-Verlag, 1999.
53. P. Molinaro, D. Roux, and O. Delfieu. Improving the calculus of the marking graph of Petri net with BDD like structure. In *Proc. of the 2nd IEEE Int. Conf. on Systems, Man and Cybernetics (SMC'02)*. IEEE Comp. Soc. Press, October 2002.
54. J. Møller, J. Lichtenberg, H. Andersen, and H. Hulgaard. Difference Decision Diagrams. In *Proc. of CSL'99*, volume 1683 of *LNCS*, pages 111–125. Springer-Verlag, 1999.
55. J. Møller, J. Lichtenberg, H. Andersen, and H. Hulgaard. Fully symbolic model checking of timed systems using Difference Decision Diagrams. In *Proc. of FLoC'99*, volume 23(2) of *ENTCS*, 1999.
56. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of the 38th Design Automation Conference (DAC'01)*, pages 530–535, June 2001.
57. P. Niebert, M. Mahfoudh, E. Asarin, M. Bozga, O. Maler, and N. Jain. Verification of Timed Automata via satisfiability checking. In *Proc. of FTRTFT'02*, volume 2469 of *LNCS*, pages 226–243. Springer-Verlag, 2002.
58. Y. Okawa and T. Yoneda. Symbolic CTL model checking of Time Petri Nets. *Electronics and Communications in Japan, Scripta Technica*, 80(4):11–20, 1997.
59. R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
60. E. Pastor, J. Cortadella, and O. Roig. Symbolic Petri net analysis using boolean manipulation. Technical Report RR-97-08, UP/DAC, Univerisitat Politécnica de Catalunya, February 1997.
61. W. Penczek and A. Półrola. Abstractions and partial order reductions for checking branching properties of Time Petri Nets. In *Proc. of ICATPN'01*, volume 2075 of *LNCS*, pages 323–342. Springer-Verlag, 2001.
62. W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.
63. W. Penczek, B. Woźna, and A. Zbrzezny. Towards bounded model checking for the universal fragment of TCTL. In *Proc. of FTRTFT'02*, volume 2469 of *LNCS*, pages 265–288. Springer-Verlag, 2002.
64. A. Półrola and W. Penczek. Minimization algorithms for Time Petri Nets. *Fundamenta Informaticae*, 2004. to appear.
65. A. Półrola, W. Penczek, and M. Szreter. Reachability analysis for Timed Automata using partitioning algorithms. *Fundamenta Informaticae*, 55(2):203–221, 2003.
66. A. Półrola, W. Penczek, and M. Szreter. Towards efficient partition refinement for checking reachability in Timed Automata. In *Proc. of FORMATS'03*, volume 2791 of *LNCS*. Springer-Verlag, 2004.
67. L. Popova and S. Marek. TINA - a tool for analyzing paths in TPNs. In *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'02)*, volume 110 of *Informatik-Berichte*, pages 195–196. Humboldt University, 1998.
68. C. Ramchandani. Analysis of asynchronous concurrent systems by timed Petri nets. Technical Report MAC-TR-120, Massachusets Institute of Technology, February 1974.

69. A. Ratzer, L. Wells, H. Lassen, M. Laursen, J. Qvortrup, M. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for editing, simulating, and analyzing Coloured Petri Nets. In *Proc. of ICATPN'03*, volume 2679 of *LNCS*, pages 450–462. Springer-Verlag, 2003.
70. Romeo: A tool for Time Petri Net analysis. http://www.irccyn.ec-nantes.fr/irccyn/d/en/equipes/TempsReel/logs, 2000.
71. S. Samolej and T. Szmuc. Modelowanie systemów czasu rzeczywistego z zastosowaniem czasowych sieci Petriego. In *Mat. IX Konf. Systemy Czasu Rzeczywistego (SCR'02)*, pages 45–54. Instytut Informatyki Politechniki Śląskiej, 2002. In Polish.
72. P. Sénac, M. Diaz, and P. de Saqui Sannes. Toward a formal specification of multimedia scenarios. *Annals of Telecommunications*, 49(5-6):297–314, 1994.
73. S. Seshia and R. Bryant. Unbounded, fully symbolic model checking of Timed Automata using boolean methods. In *Proc. of CAV'03*, volume 2725 of *LNCS*, pages 154–166. Springer-Verlag, 2003.
74. J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proc. of STACS'96*, volume 1046 of *LNCS*, pages 347–359. Springer-Verlag, 1996.
75. M. Sorea. Bounded model checking for Timed Automata. In *Proc. of MTCS'02*, volume 68(5) of *ENTCS*. Elsevier Science Publishers, 2002.
76. R. L. Spelberg, H. Toetenel, and M. Ammerlaan. Partition refinement in real-time model checking. In *Proc. of FTRTFT'98*, volume 1486 of *LNCS*, pages 143–157. Springer-Verlag, 1998.
77. K. Strehl and L. Thiele. Interval diagram techniques for symbolic model checking of Petri nets. In *Proc. of DATE'99*, LNCS, pages 756–757. IEEE Comp. Soc. Press, 1999.
78. O. Strichman, S. Seshia, and R. Bryant. Deciding separation formulas with SAT. In *Proc. of CAV'02*, volume 2404 of *LNCS*, pages 209–222. Springer-Verlag, 2002.
79. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.
80. J. Tsai, S. Yang, and Y. Chang. Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications. *IEEE Trans. on Software Eng.*, 21(1):32–49, 1995.
81. W. van der Aalst. Interval timed coloured Petri nets and their analysis. In *Proc. of ICATPN'93*, volume 961 of *LNCS*, pages 452–472. Springer-Verlag, 1993.
82. I. B. Virbitskaite and E. A. Pokozy. A partial order method for the verification of Time Petri Nets. In *Fundamental of Computation Theory*, volume 1684 of *LNCS*, pages 547–558. Springer-Verlag, 1999.
83. B. Walter. Timed Petri nets for modelling and analysing protocols with real-time characteristics. In *Proc. of the 3rd IFIP Workshop on Protocol Specification, Testing, and Verification*, pages 149–159. North Holland, 1983.
84. F. Wang. Region Encoding Diagram for fully symbolic verification of real-time systems. In *Proc. of the 24th Int. Computer Software and Applications Conf. (COMPSAC'00)*, pages 509–515. IEEE Comp. Soc. Press, October 2000.
85. F. Wang. Verification of Timed Automata with BDD-like data structures. In *Proc. of VMCAI'03*, volume 2575 of *LNCS*, pages 189–205. Springer-Verlag, 2003.
86. B. Woźna, A. Zbrzezny, and W. Penczek. Checking reachability properties for Timed Automata via SAT. *Fundamenta Informaticae*, 55(2):223–241, 2003.
87. M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In *Proc. of CAV'93*, volume 697 of *LNCS*, pages 210–224. Springer-Verlag, 1993.

88. W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *Proc. of the 7th IFIP WG6.1 Int. Conf. on Formal Description Techniques (FORTE'94)*, volume 6 of *IFIP Conference Proceedings*, pages 243–258. Chapman & Hall, 1994.
89. T. Yoneda and H. Ryuba. CTL model checking of Time Petri Nets using geometric regions. *IEICE Trans. Inf. and Syst.*, 3:1–10, 1998.
90. T. Yoneda and B. H. Schlingloff. Efficient verification of parallel real-time systems. *Formal Methods in System Design*, 11(2):197–215, 1997.
91. S. Yovine. KRONOS: A verification tool for real-time systems. *Springer International Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, 1997.
92. A. Zbrzezny. Improvements in SAT-based reachability analysis for Timed Automata. *Fundamenta Informaticae*, 2004. to appear.
93. L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proc. of Int. Conf. on Computer-Aided Design (ICCAD'01)*, pages 279–285, 2001.