Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cosrev

Survey

A survey of timed automata for the development of real-time systems

*Md Tawhid Bin Waez**, *Juergen Dingel*, *Karen Rudie*

Queen's University, ON, Canada

ARTICLE INFO

Article history:
Received 12 October 2011
Received in revised form
30 May 2013
Accepted 30 May 2013

Keywords:
Timed automata
Survey
Real-time systems
Formal models
Semantics
Timed regular languages
Decision problems
Variants
Implementability
Tools

ABSTRACT

Timed automata are a popular formalism to model real-time systems. They were introduced two decades ago to support formal verification. Since then they have also been used for other purposes and a large number of variants has been introduced to be able to deal with the many different kinds of requirements of real-time system development. This survey attempts to introduce a massive and complicated theoretical research area to a reader in an easy and compact manner. One objective of this paper is to inform a reader about the theoretical properties (or capabilities) of timed automata which are (or might be) useful for real-time model driven development. To achieve this goal, this paper presents a survey on semantics, decision problems, and variants of timed automata. The other objective of this paper is to inform a reader about the current state of the art of timed automata in practice. To achieve the second aim, this article presents a survey on timed automata's implementability and tools.

Contents

* *Corresponding author.* Tel.: +1 3433332288.
E-mail addresses: waez@cs.queensu.ca (M.T.B. Waez), dingel@cs.queensu.ca (J. Dingel), karen.rudie@queensu.ca (K. Rudie).

1574-0137/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.cosrev.2013.05.001

1. Introduction

Timed automata (TA) [1,2] were introduced by Alur and Dill in the early 1990s. Since then, TA have become one of the most dominant formal models to support *Model Driven Development (MDD)* of real-time systems. A real-time transition system of a timed automaton can be infinitely large due to its ability to express dense time. A real-time transition system can be converted into an equivalent finitely large symbolic (real-time) transition system called *region graph* where reachability is decidable. Decidability of reachability is a core requirement for automated formal verification and this property of TA plays a foremost role to establish TA as the major real-time formal model. Later on, *zone graphs* were developed and evolved to provide better scalability in practice compared to region graphs. Rich closure properties and decidability of many important decision problems have contributed to the adaptation of TA in many approaches to support MDD of real-time systems. During the first two decades of TA, many kinds of generalizations and variants of timed automata have been proposed and studied to address practically all aspects and features of real-time systems. The strong foundation of TA has inspired the emergence of a huge number of tools for analysis, verification, controller synthesis, and code synthesis for TA. This survey is an attempt to provide an organized description of the development of TA and its variants from theory to practice during the first two decades after the birth of TA.

This survey presents a compact discussion on syntax, operational semantics, and two major symbolic semantics, named region and zone, of TA. The survey describes different kinds of analysis techniques such as region-based, zone-based, and flattening-based techniques. Decision problems and closure properties for TA are also listed in this survey. This paper classifies eighty variants of timed automata in an effort to determine current developments. It uses analysis techniques, formal properties, and decision problems to draw distinctions between different versions. Moreover, the paper discusses the challenges behind using a TA specification to derive an implementation of a working real-time system and presents some solutions. Finally, the paper lists and classifies forty tools supporting TA. We attempt to help the reader to navigate the vast literature in the field of TA, to highlight closure properties, decision problems, differences, and similarities, and to reveal research trends and promising avenues for future exploration.

This paper includes only those theorems which are important to describe the words surveyed. The paper does not provide any proof or proof sketch. The remainder of the paper is organized as follows: Section 2 discusses the syntax of TA, while Section 3 explains the operational and symbolic semantics of TA. Section 4 presents formal linguistic aspects of TA. Section 5 enumerates eighty variants of TA and then classifies them into eleven classes. Section 6 discusses implementability challenges and solutions. Section 7 presents some academic tools which are based on TA. The paper concludes in Section 8.

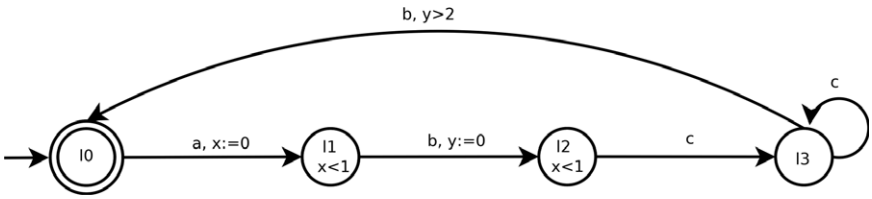


Fig. 1 – A timed automaton with 2 clocks [4].

2. Syntax

A timed automaton is a finite state automaton with a set of asynchronous (nonnegative real valued) clocks and a set of clock constraints. A *clock valuation* over the set of clocks is a mapping which assigns to each clock a nonnegative real value. An *initial clock valuation* maps each clock of a timed automaton to zero. A vertex in a timed automaton is called a *location*. A location is associated with a clock constraint called the *local invariant*[1] of that location. Control can stay in a location only if the clock valuation satisfies the local invariant of that location. Local invariants are used to ensure the progress of the model [3], that is, control cannot stay in a location forever. Instead of local invariants, *Büchi* or *Muller* acceptance conditions can be used to enforce progress [1,2]. An edge in a timed automaton is called a *switch*. A switch is associated with a clock constraint, a subset of the clocks, and a *label* (with a symbol). A clock constraint which is associated with a switch is called the *guard* of that switch. A switch can be taken only if the clock valuation satisfies the guard of that switch. Guards are used to restrict the behavior of the automaton. Each associated clock of a switch is reset to 0 when the switch occurs. At any instant, the value of a clock equals the time elapsed since the last time it was reset. While switches are instantaneous, time can elapse in a location. Consider the example [4] in Fig. 1 with two clocks (x and y). The clock x is set to 0 each time the system switches from $l0$ to $l1$ on symbol a. The local invariant $(x < 1)$ associated with the locations $l1$ and $l2$ ensures that the c-labeled switch from $l2$ to $l3$ happens within one time unit of the occurrence of a. Resetting clock y together with the b-labeled switch from $l1$ to $l2$ and the guard of the d-labeled switch from $l3$ to $l0$ ensures that the delay between b and the following d is always greater than two time units.

Definition 1. A timed automaton A is a tuple $\langle L, L_0, L_F, \Sigma, \mathcal{C}, E, I\rangle$, where L is a finite set of locations, $L_0 \subseteq L$ is the set of initial locations, $L_F \subseteq L$ is the set of final locations, Σ is a finite alphabet, $\mathcal{C}$ is a finite set of nonnegative real valued clocks, $E \in L \times \Phi(\mathcal{C}) \times (\Sigma \cup \{\epsilon\}) \times 2^{\mathcal{C}} \times L$ is the set of switches (edges), and $I : L \longrightarrow \Phi(\mathcal{C})$ is a mapping that assigns local invariants to locations.

The set $\Phi(\mathcal{C})$ of clock constraints δ is defined inductively by $\delta := x \sim q \mid x - y \sim q \mid \neg\delta \mid \delta_1 \wedge \delta_2 \mid true$ and $q \in \mathbb{Q}$, $\sim \in \{=, <, >, \leq, \geq\}$, elements of the alphabet Σ are observable actions, ϵ represents unobservable actions, and $\mathcal{C}$ is ranged over by x, y etc. The above stated clock constraints only allow one to compare a clock or the difference of two clocks with a rational constant. Clock constraints of the form of $x-y \sim q$ are called *diagonal clock constraints* or *difference clock constraints*. A timed automaton without diagonal clock constraints is called a *diagonal-free TA* [5]. A *k-bounded clock constraint* is a clock constraint which involves only constants between $-k$ and k.

A switch $e = \langle l, a, \phi, \gamma, l'\rangle \in E$ from location l to l' can occur and reset the set of clocks $\gamma \in 2^{\mathcal{C}}$ on symbol a if the current clock valuation ν satisfies the guard $\phi(\nu \vDash \phi)$. Only the clock constraints which are downwards closed[2] are used as local invariants because a local invariant merely asserts how long control can stay in the associated location of that local invariant.

3. Semantics

3.1. Operational semantics

Definition 2. A *timed transition system* is a tuple $\langle S, S_0, S_F, \Sigma \cup \mathbb{R}_+, \rightarrow\rangle$ where S is a set of states, $S_0 \subseteq S$ is a set of initial states, $S_F \subseteq S$ is a set of final states, Σ is an alphabet, and $\rightarrow \subseteq S \times (\Sigma \cup \{\epsilon\} \cup \mathbb{R}_+) \times S$.

Definition 3. The *semantics of a timed automaton* $A = \langle L, L_0, L_F, \Sigma, \mathcal{C}, E, I\rangle$ is defined by associating a timed transition system $TS(A)$ of the same alphabet with A [1,2]: a state in $TS(A)$ is expressed as a pair (l, ν) such that $l \in L$ and ν is a clock valuation (for $\mathcal{C}$) which satisfies the local invariant $I(l)$. A pair (l, ν) is in S_0 iff l is an initial location ($l \in L_0$) and ν is the initial clock valuation ν_0. Similarly, a state (l, ν) is a final state iff l is a final location ($l \in L_F$). $TS(A)$ can have two types of transitions:

Action transition: $(l, \nu) \xrightarrow{a} (l', \nu[\gamma := 0])$ for a switch $\langle l, a, \phi, \gamma, l'\rangle$ if $\nu \vDash \phi$, where $a \in \{\Sigma \cup \{\epsilon\}\}$, $\gamma \in 2^{\mathcal{C}}$ and $\nu[\gamma := 0]$ denotes a clock valuation that differs from ν only in that clocks in γ have been reset to 0.

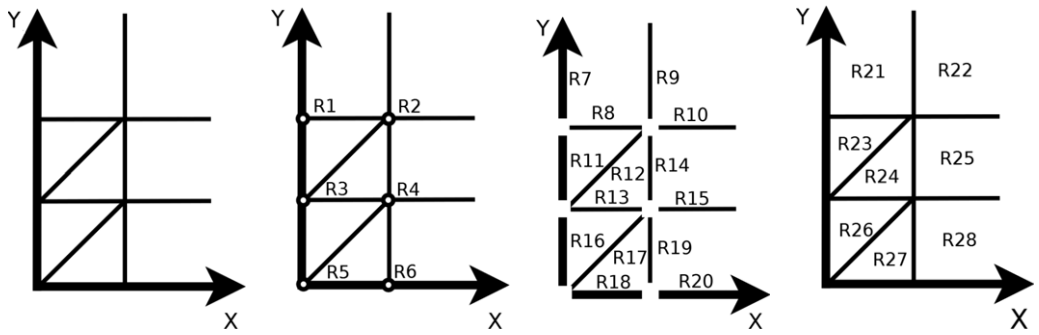
Time transition: $(l, \nu) \xrightarrow{\tau} (l, \nu + \tau)$ if $(\nu + \tau') \vDash I(l)$ for $\forall\tau' : 0 \leq \tau' \leq \tau$, where $\tau \in \mathbb{R}_+$.

Due to the real-value time transitions, the state-space of the timed transition system of a timed automaton could be infinitely large.

A *timed action* is a pair (t, a), where action $a \in (\Sigma \cup \{\epsilon\})$ is taken by a timed automaton A after $t \in \mathbb{R}_+$ time units since A has been started. The absolute time t is called a *time-stamp*

[1] A timed automaton with local invariants is called *safety TA* [3].

[2] A clock constraint in the form $x \preceq n$ or $x - y \preceq n$ is downwards closed, where $\preceq \in \{<, \leq\}$ and n is a nonnegative integer.



(a) Clock regions. **(b) 6 intersections.** **(c) 14 lines.** **(d) 8 spaces.**

Fig. 2 – All the 28 clock regions in Fig. 2(a) for the TA of Fig. 1: 6 intersections, 14 lines, and 8 spaces.

of the action a. A *timed word* is a sequence of timed actions $\xi = (t_1, a_1)(t_2, a_2)...(t_i, a_i)$ where $t_i \leq t_{i+1}$ for $\forall i : i \geq 1$. A *run* of A in TS with initial state $\langle l_0, \nu_0\rangle$ over the timed word $\xi = (t_1, a_1)(t_2, a_2)...(t_i, a_i)$ is a sequence of transitions:

$$\langle l_0, \nu_0\rangle \xrightarrow{t_1} \langle l_0, \nu_0'\rangle \xrightarrow{a_1} \langle l_1, \nu_1\rangle \xrightarrow{t_2-t_1} \langle l_1, \nu_1'\rangle \xrightarrow{a_2} \langle l_2, \nu_2\rangle \cdots \xrightarrow{a_i} \langle l_i, \nu_i\rangle.$$

A run is *accepting iff* $\langle l_i, \nu_i\rangle$ is a final state. The *timed language* Σ_t^* over Σ is the set of all timed words over Σ. The *generated timed language* $L_{gt}(A) \subseteq \Sigma_t^*$ is the set of all timed words for which there exists a run of TA A. The set of all timed words with an accepting run of a timed automaton A is the *accepted timed language* $L_t(A) \subseteq L_{gt}(A)$ by A. The *untimed language* L_u is the set of all words in the form $a_1a_2a_3...$ for which there exists a timed word $\xi = (t_1, a_1)(t_2, a_2)\cdots(t_i, a_i) \in \Sigma_t^*$.

3.2. Symbolic semantics

Exhaustive verification via state-space exploration is not possible on an infinitely large state-space. In the last two decades, researchers have made many attempts to convert this infinite state-space into an abstract state-space with a finite, tractable number of states such that this coarser state-space preserves all the important properties (for modeling and verification) of the original state-space.

3.2.1. Region graph

An infinite state-space of a timed transition system $TS(A)$ can be converted into an equivalent finite state-space of a symbolic transition system called a *region graph* $\mathcal{R}(A)$ [6,1,2]. The decidability results (e.g., reachability analysis, untimed language inclusion, language emptiness, etc.) in TA are based on this notion of a symbolic state-space.

Definition 4. A *region*, a state of a region graph $\mathcal{R}(A)$, is a pair $\langle l, r\rangle$; where l is a location and r is a set of clock valuations known as clock region. Two clock valuations ν and μ are in the same clock region, denoted $\nu \approx^{\mathcal{R}} \mu$, if for any clock x_1 these clock valuations have equal integral part ($\lfloor \nu(x_1) \rfloor = \lfloor \mu(x_1) \rfloor$) and for all clocks these clock valuations preserve the order of the fractional parts (if $fr(\nu(x_1)) \leq fr(\nu(x_i))$ then $fr(\mu(x_1)) \leq fr(\mu(x_i))$, where $fr(c) = c - \lfloor c \rfloor$ and $i \in \mathbb{N}$).

The integral part of a clock value is important to decide whether or not a specific clock constraint is satisfied, while the ordering of the fractional parts is needed to decide which clock will change its integral part first.

Theorem 1. *If the number of clocks $|\mathcal{C}|$ is fixed and each clock $x \in \mathcal{C}$ has a maximal constant m_x, then the number of clock regions is finite: the number of clock regions can be at most $|\mathcal{C}|! \cdot 4^{|\mathcal{C}|} \cdot \prod_{x\in\mathcal{C}}(m_x + 1)$* [2].

All clock regions for the TA of Fig. 1 are shown in Fig. 2.

Theorem 2. *If $\nu \approx^{\mathcal{R}} \mu$ then $\langle l, \nu\rangle$ and $\langle l, \mu\rangle$ are* untimed bisimilar *(or bisimilar w.r.t. $L_u(A)$) for $\forall l : l \in L$* [2].

As a consequence, untimed bisimulation is used to construct the $\mathcal{R}(A)$. The first attempt to construct region graphs was made on diagonal-free TA. Diagonal clock constraints are necessary to model many applications such as scheduling problems [7]. It was shown that a timed automaton A with difference clock constraints can be converted into an equivalent TA A' which has no difference clock constraints [5]. This conversion is based on a region construction. The size of the transformed model is exponential in the number of diagonal clock constraints.

The number of clock regions in $\mathcal{R}(A)$ grows exponentially with the number of clocks and the size of maximal constants in the clock constraints. Many techniques for the minimization of region automata have been proposed [8,3,9]. None of these proposed techniques has been successful in practice. Region automata are not used in practice because the number of regions is often too large to be explored exhaustively.

3.2.2. Zone graph

A practically efficient abstract state-space of a timed automaton A is given by its Zone Graph $\mathcal{Z}(A)$ [10].

Definition 5. A *zone* $\langle l, [\delta]\rangle$ is a pair of a location l and a *clock zone* $[\delta]$ which is the maximal set of clock valuations satisfying $\delta \in \Phi(\mathcal{C})$.

If a timed automaton has n clocks, then its clock zones are convex sets in n-dimensional euclidean space.

Theorem 3. *Every clock region is a clock zone* [11].

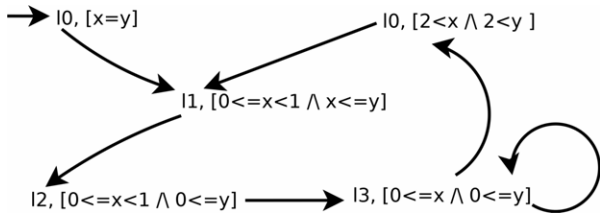


Fig. 3 – Zone graph for the TA of Fig. 1 with only 5 zones.

Theorem 4. *If the addition of two clock regions (or clock zones) is a convex set then the addition is a clock zone* [11].

Theorem 5. *The number of clock zones is the number of convex unions of clock regions* [12].

In the worst case, this number is exponential in the number of clock regions. In practice, clock zones are coarser and more compact than clock regions (e.g., the TA of Fig. 1 has 28 clock regions as shown in Fig. 2, while it has only 5 clock zones as shown in Fig. 3). Zones have been used to implement all the major TA-based tools (e.g., UPPAAL [13], KRONOS [14]).

For a timed automaton $A = \langle L, L_0, L_F, \Sigma, \mathcal{C}, E, I\rangle$, its zone graph $\mathcal{Z}(A)$ is a transition system: states of $\mathcal{Z}(A)$ are zones of A, the zone $\langle l_0, [\mathcal{C} = 0]\rangle$ is the initial state of $\mathcal{Z}(A)$ (where $l_0 \in L_0$ and $\mathcal{C} = 0$ means that the value of any clock in $\mathcal{C}$ is 0), and for every switch $e = \langle l, a, \phi, \gamma, l'\rangle \in E$ and every zone $\langle l, [\delta]\rangle$ there is a transition $\langle\langle l, [\delta]\rangle, a, succ_e(\langle l, [\delta]\rangle)\rangle$; where $succ_e$ is a successor function which returns all the zones which can be reached from the zone $\langle l, [\delta]\rangle$ by first performing the switch e, then letting time pass in the new location, while continuously satisfying the local invariant. The successor function $succ_e$ and reachability analysis in a zone graph are possible because clock zones are closed under the three operations $[\delta_1] \wedge [\delta_2]$, $[\delta]^{\Uparrow,\tau}$, and $[\delta][\gamma := 0]$ where $[\delta_1] \wedge [\delta_2]$ denotes the intersection of $[\delta_1]$ and $[\delta_2]$, $[\delta]^{\Uparrow,\tau}$ denotes the set of interpretations for $\nu + \tau$ for $\nu \in [\delta]$ and $\tau \in \mathbb{R}_+$, and $[\delta][\gamma := 0]$ denotes the set of clock valuations $\nu[\gamma := 0]$ for $\nu \in [\delta]$ and $\gamma \in \mathcal{C}$.

A clock zone $[\delta]$ is *closed under entailment*, if δ cannot be strengthened[3] without reducing the solution set. A *canonical zone graph* $\mathcal{Z}(A)$ means that for every $[\delta] \in \mathcal{Z}(A)$, there is a unique clock zone $[\delta']$ (where $\delta' \in \Phi(\mathbb{C})$) such that $[\delta]$ and $[\delta']$ have exactly same solution set and $[\delta']$ is closed under entailment. Clock zones of a canonical zone graph are represented and manipulated in a data structure called *Difference Bounded Matrices* (DBM) [15,16,10]. It is the major structure for the efficient implementation of real-time state-space exploration using symbolic semantics.

Zone graphs are not always finite [17,18], which makes exhaustive exploration impossible. To remedy this problem, one approach is to construct a *region-closed zone graph* [19–21]: replace each $[\delta] \in \mathcal{Z}(A)$ by the union of the regions of $\mathcal{R}(A)$ which intersect $[\delta]$. Since the number of regions is finite, there is a finite number of zones after this operation. The region closure of a zone may not be convex. As a result, DBM cannot be used. For this reason, the region-closed zone graph is not used in practice.

Another approach to guarantee finiteness of zone graph is the use of an abstraction operator called the *k-extrapolation* (k is a constant supposed to be greater than the maximal constant occurring in A) [17,18,22]. The k-extrapolation operator abstracts $\mathcal{Z}(A)$ into another zone graph $\mathcal{Z}'(A)$, denoted *k-extrapolated zone graph*, such that all constraints defined in $\mathcal{Z}'(A)$ are k-bounded. The k-extrapolated zone graph is finite, since the number of clock zones with bounded constraints is finite. As an example, a finite k-extrapolated zone graph of the infinite zone graph of Fig. 4(b) is shown in Fig. 4(c).

Theorem 6. *A k-extrapolated zone graph is* correct for reachability[4] *only for diagonal-free TA* [23,24].

If A has any diagonal constraint, then $\mathcal{Z}(A)$ may have a reachable zone $\langle l, [\delta]\rangle$ where l is not a reachable location in A. A k-extrapolated zone graph (with diagonal constraints) is correct for reachability [17], if clock valuation ν satisfies a diagonal constraint δ *iff* clock valuation μ satisfies δ where ν and μ are k-extrapolated zone equivalent clock valuations. One method to ensure correctness for reachability of a k-extrapolated zone graph is to check this property [17]. However, checking this property may suffer from an exponential blow-up in the number of zones. The number of zones is multiplied by 2^n, where n is the number of diagonal constraints. To remedy this problem, a new method has been proposed based on counter-example[5] guided abstraction refinement [25] and has been applied [26] in UPPAAL [13]. Not all the diagonal constraints cause incorrectness for reachability. In practice, diagonal constraints produce an incorrect result only very rarely. Since counter-examples are very rare, this refinement method causes very little overhead in practice.

4. Timed regular languages and the decision problems

Definition 6. A language $L \subseteq \Sigma_t^*$ is a *timed regular language*, if there exists a timed automaton A such that $L = L_t(A)$.

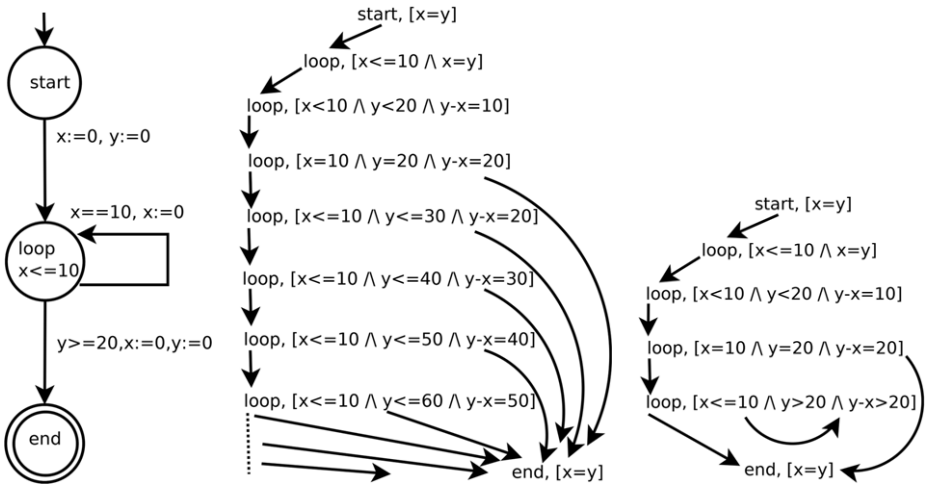
Theorem 7. *An untimed language of a timed regular language is a* regular language [2].

Timed regular expressions [27] can be used to represent timed regular languages and operations on them. Some variants [28,29] of timed regular expressions exist in the literature. In MDD, closure properties and decision problems are crucial for modeling, operations on models, and formal verification of a model. For example, the underlying languages need to be closed under *intersection* and *shuffle* to model a concurrent system using an synchronous and interleaving semantics, while *emptiness checking* is used to detect the violation of *safety properties* ("nothing bad will happen") in a model.

[3] Let $\delta_1 = \delta \wedge x \preceq n_1$ and $\delta_2 = \delta \wedge x \preceq n_2$ are two clock constraints such that δ_1 and δ_2 have the same solution set and $n_1 > n_2$, then δ_1 can be strengthened by replacing n_1 by n_2 in δ_1.

[4] We say that a symbolic transition system T' of an original transition system T is correct for reachability *iff* a state s is reachable in T then there is a reachable symbolic state s' in T' which contains s.

[5] A counter-example is a trace where l in A is not reachable, but $\langle l, [\delta]\rangle$ in $\mathcal{Z}(A)$ is reachable.



(a) A timed automation. **(b) An infinite zone graph.** **(c) A k-extrapolated zone graph.**

Fig. 4 – A timed automaton with its infinite zone graph and its k-extrapolated (here, $k = 20$) zone graph [17].

Table 1 – Closure properties of TA.

Property	Closure under	Property	Closure under
Union	Yes	Kleene-star	Yes
Intersection	Yes	Projection	Yes
Concatenation	Yes	Shuffle	No
Renaming	Yes	Complementation	No

Theorem 8. *Timed regular languages are closed under* union [1,2], intersection [1,2], concatenation [27], projection [1], renaming [1], *and* Kleene-star [27].

Theorem 9. *Timed regular languages are not closed under* complementation [1,2] *and* shuffle [30,31].

Closure properties of TA are summarized in Table 1.

Theorem 10. *The* emptiness checking *problem for TA is PSPACE-complete and can be solved in time* $O(|E| \cdot |\mathcal{C}|! \cdot 4^{|\mathcal{C}|} \cdot (m \cdot m' + 1)^{|\mathcal{C}|})$ *where* m *is the largest numerator in the constants in the clock constraints and* m' *is the least-common-multiple of the denominators of all the constants in the clock constraints* [2,32].

Theorem 11. Minimum – time reachability [6] *for TA is PSPACE-hard* [33,34].

Theorem 12. Timed bisimulation [35–37] *and* timed simulation [38] *are decidable in* EXPTIME.

Theorem 13. Universality [2], language equivalence [1,2], language inclusion [1,2], determinizability[7] [39,40], computing the clock degree, [40,230], minimization of the number of clocks[8] [39,40], *and* reducing the size of constants[9] [40] *for timed automata are* undecidable.

A *flat* TA is a timed automaton which does not have any nested loops: for every location l there is at most one non-empty path from l to itself.

Theorem 14. *Any TA can be emulated by a flat TA* [41].

Comon and Jurski [41] have shown that the *binary reachability* between any two (set of) states of a timed transition system of a timed automaton is decidable and they left the complexity issue as an open problem. Instead of the conventional region-based (or zone-based) technique, they first convert a timed automaton to an equivalent flat TA and then use the *additive theory of real numbers* to prove the decidability of binary reachability in a timed automaton. We will call their technique *flattening technique*. The flattening technique allows one to express and verify some important properties that cannot be expressed or verified by region-based (or zone-based) techniques such as "the delay between event $a1$ and event $b1$ is never larger than twice the delay between even $a2$ and event $b2$". On the other hand, their technique is unable to express all the region-based (or zone-based) timing properties, e.g., the flattening technique unable to express unavoidability. These decision problems are summarized in Table 2.

Definition 7. A *deterministic* TA has at most one initial state, no ϵ-transitions, and no pair of switches which have the same action from the same source location with a common clock valuation which can satisfy the guards of both switches.

A deterministic TA has only one run.

[6] Given a timed automaton A, is there a run of A from some initial location $l_0 \in L_0$ to some final location $l_f \in L_f$? If so, find such a run which consumes minimum-time.

[7] Given an automaton A, does there exists a deterministic automaton B such that $L(A) = L(B)$? If so, construct B.

[8] Given a timed automaton A with n clocks, does there exists a timed automaton B with $n-1$ clocks, such that $L_t(A) = L_t(B)$? If so, construct B.

[9] Given a timed automaton A where constants are not greater than k, does there exist a timed automaton B where constants are not greater than $k-1$, such that $L_t(B) = L_t(A)$? If so, construct B.

Table 2 – Complexity of decision problems for TA.

Problem	Complexity	Problem	Complexity
Emptiness checking	PSPACE-complete	Minimum-time reachability	PSPACE-hard
Timed bisimulation	EXPTIME	Computing the clock degree	Undecidable
Timed simulation	EXPTIME	Language equivalence	Undecidable
Universality	Undecidable	Reducing the size of constants	Undecidable
Language inclusion	Undecidable	Minimiza. of the number of clocks	Undecidable
Determinizability	Undecidable	Binary reachability	Decidable

Table 3 – Closure properties of deterministic TA.

Property	Closed under	Property	Closed under
Union	Yes	Complementation	Yes
Intersection	Yes	Projection	No
Renaming	No		

Table 4 – Complexity of decision problems for deterministic TA.

Problem	Complexity	Problem	Complexity
Emptiness checking	PSPACE-complete	Language inclusion	PSPACE-complete
Universality	PSPACE-complete	Language equivalence	PSPACE-complete

Theorem 15. *Deterministic TA are* strictly contained *in (nondeterministic) TA* [1,2].

Theorem 16. *Deterministic TA are closed under* union [1,2], intersection [1,2], *and* complement [1,2].

Theorem 17. *Deterministic TA are not closed under* projection [1,2] *and* renaming [42].

These closure properties of deterministic TA are summarized in Table 3.

Theorem 18. Emptiness checking, universality, language inclusion, languages equivalence *problems for deterministic TA are* PSPACE-*complete* [1,2].

These decision problems of deterministic TA are summarized in Table 4.

5. Variants of TA

Many variants of TA have been proposed in the literature. There are three primary motivations behind this flourish of variants: the first and most significant one is to improve existing analysis capabilities of TA for the modeling of real-time systems (e.g., to find optimal paths [43], and check schedulability [44,45], and check memory consumption [43,46,45], etc.); the second one is to increase expressiveness by adding features such as probability [47,48] or recursion [49]; and the last reason is to increase conciseness of the model [50]. There are also some variants of the semantics [51,52] to make TA a more robust and accurate real-time model.

This paper lists almost eighty variants of TA and there may be many more. The number is surprising if one considers that the first variant was proposed only two decades ago. The paper classifies all these variants into eleven classes: *classical timed automata* (Section 5.1), *TA with other clock constraints* (Section 5.2), *TA with other clock updates* (Section 5.3), *TA with other clock rates* (Section 5.4), *TA with resources* (Section 5.5), *TA with probability* (Section 5.6), *TA with communication* (Section 5.7), *TA with determinizability* (Section 5.8), *TA with self-embedded recursion* (Section 5.9), *TA with succinctness* (Section 5.10), and *TA with games* (Section 5.11). This classification (Table 5) is intended to help a reader to understand the major objectives, similarities, and dissimilarities of a huge number of variants of a complicated theoretical research area. According to Table 5, the class of TA with resources has the highest number of variants and the class of TA with determinizability has the second highest number of variants. The main motivation behind the flourish of the class of TA with resources is to improve expressiveness and analysis capabilities. On the other hand, the goal of the research on TA with determinizability is to improve the complexity of key decision problems and to achieve more closure properties. Typically, an increase in expressive power and analysis capabilities comes at the expense of increased complexity and fewer closure properties. Both of these conflicting goals are being extensively researched. This section attempts to provide a glimpse into all these eleven classes by discussing the fewest possible variants with their major decidability results and tools.

5.1. *Classical TA*

We classify TA variants which have the same (major) theoretical properties and expressive power of the *standard TA* (Definition 1) as *classical* TA. Either a construction rule or an accepting condition of a variant of classical TA is dissimilar to other variants of this class. Büchi TA [1], Muller TA [1], diagonal-free TA [1], TA with diagonal constraints [1], TA with ϵ-transitions [1], TA without ϵ-transitions [1], safety TA [3], and flat TA [41] are the major variants of this class.

5.2. *TA with other clock constraints*

This subsection presents variants of TA which add more expressive clock constraints with the existing clock constrains of classical TA. Most of them lose many important theoretical properties to facilitate extra expressiveness. In terms of practical applications *parametric TA* [56] is the most influential and important class of variants in this subsection.

5.2.1. *TA with periodic clock constraints*

Theorem 19. *The class of ϵ-transition-free TA is* strictly less expressive *than the class of TA with ϵ-transitions* [5].

The TA in Fig. 5 accepts a timed language L_ϵ which can be described as follows: in each open time interval $(i, i+1), i \geq 0$ there occurs at most one b; moreover, there is an a at time $i+1$ if and only if there is no b in $(i, i+1)$. This L_ϵ cannot be accepted by a timed automaton which has no ϵ-transitions.

Table 5 – Classification of the variants of TA.

Class	Variants
Classical TA	Büchi TA [1], Muller TA [1], Diagonal-Free TA [1], TA with Diagonal Constraints [1], TA with ϵ-Transitions [1], TA without ϵ-Transitions [1], Safety TA [3], Flat TA [41]
TA with Other Clock Constraints	TA with Multiplication Clock Constraints [1], TA with Periodic Clock Constraints [53], TA with Additive Clock Constraints [54], TA with Irrational Clock Constraints [55], Parametric TA [56], L/U Automata [57], TA with ASAP Semantics [52]
TA with Other Clock Updates	Updatable TA [50], Suspension Automata [58], Integer Reset TA [59], Weighted Integer Reset TA [60], Task Automata [45], Fixed Task Automata [45], Flexible Task Automata [45], Feedback Task Automata [45], Non-Feedback Task Automata [45],
TA with Other Clock Rates	Hybrid Automata [61], Rectangular Automata [62], Controlled TA [63], Stopwatch Automata [64], Distributed Time-Asynchronous Automata [65], Distributed TA with Independently Evolving Clocks [66], Interrupt TA [67], Robust TA [68], Perturbed TA [69]
TA with Resources	Weighted TA [43], Priced TA [46], Uniformly-Priced TA [70], Dual-Priced TA [71], Multi-Priced TA [71], Priced Probabilistic TA [72], Extended TA with Tasks [73], Extended TA with Asynchronous Processes [7], Task Automata [45], Fixed Task Automata [45], Flexible Task Automata [45], Feedback Task Automata [45], Non-Feedback Task Automata [45], Concavely-Priced TA [74], Concavely-Priced Probabilistic TA [75], Timed P Automata [76], Weighted Integer Reset TA [60], Priced Timed Game Automata [77]
TA with Probability	Discrete Probabilistic TA [78], Continuous Probabilistic TA [79], Concavely-Priced Probabilistic TA [75], First-Order Probabilistic TA [80]
TA with Communication	Communicating TA [81], Communicating Hierarchical TA [82], Multi-Queue Discrete TA [83], Omega Deterministic Timed Alternating Finite Automata [84], Synchronized Concurrent TA [85], Queue-Connected Discrete TA [86], Phase Event Automata [87], Timed Cooperating Automata [88], Cottbus TA [89]
TA with Determinizability	Event-Clock Automata [42], Event-Recording Automata [42], Event-Predicting Automata [42], Eventual TA [90], Recursive Event-Clock Automata [91], Product Interval TA [92], TA with Input-Determined Guards [93], Continuous TA with Input-Determined Guards [94], Counter-Free Input-Determined TA [95], Event-Clock Visibly Pushdown Automata [96]
TA with Self-Embedded Recursion	Recursive Event-Clock Automata [91], Discrete Pushdown TA [49], Pushdown TA [49], Past Pushdown TA [97], Timed Visibly Pushdown Automata [98], Event-Clock Visibly Pushdown Automata [96], Recursive TA [99], Timed Recursive State Machines [100]
TA with Succinctness	TA with Deadlines [101], Prioritized TA [102], Variable Driven TA [103], TA with Urgent Transitions [104], Alternating TA [105], Weak Alternating TA [106]
TA with Games	Timed Game Automata [107], Priced Timed Game Automata [77]

Theorem 20. *ϵ-transitions* without resets *can be removed from a timed automaton* [108] *and an ϵ-transition which* does not lie in a loop *can be eliminated* [109].

Periodic clock constraints are clock constraints of the form $d+n\cdot\theta \le x \le e+n\cdot\theta$ or $d+n\cdot\theta \le x-y \le e+n\cdot\theta$ where $n \in \mathbb{N}$, $e \in \mathbb{R}$, and $\theta \in \mathbb{R}_+$. Periodic clock constraints can express properties such as "the value of clock x is odd" or "the value of clock x is of the form $0.7 + 4 \cdot n$ where n is some integer".

Theorem 21. TA with periodic clock constraints *in the guards and classical TA have the same expressive power* [53].

Theorem 22. *The ϵ-transition-free (deterministic) TA with periodic clock constraints in the guards are* strictly more expressive *than the ϵ-transition-free classical (deterministic) TA* [53].

Theorem 23. *All ϵ-transitions can be removed from TA by using periodic clock constraints and periodic clock updates*[10] [110].

5.2.2. *Additive, multiplication, and irrational clock constraints* Clock constraints of the form of $x + y \sim q$ are called *additive clock constraints*.

Theorem 24. *The emptiness checking problem is undecidable for* TA with additive clock constraints *which have four clocks* [54].

[10] During a periodic update of a clock that clock is reset to a periodic value instead of 0.

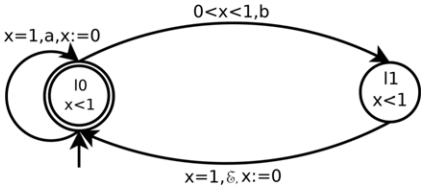


Fig. 5 – A TA with an ϵ-transition which has no equivalent ϵ-transition-free TA [5].

TA with additive clock constraints having two clocks are strictly more expressive than classical TA with two clocks.

Theorem 25. *The emptiness checking problem is decidable for TA with additive clock constraints having two clocks* [54].

While the emptiness checking problem is still open for TA with additive clock constraints which have three clocks.

Theorem 26. *Introducing clock constraints such as $x = q \cdot y$ in the guards makes the emptiness checking problem for TA undecidable* [2].

Theorem 27. *Allowing irrational constants in the clock constraints causes the emptiness checking problem to be undecidable* [55].

A summary of all kinds of clock constraints and their effect on the complexity of reachability checking is shown in Table 6.

Table 6 – Complexity of reachability checking using different clock constraints.

Clock constraint	Reachability	Clock constraint	Reachability
$x \sim q$	PSPACE-complete	$d + n \cdot \theta \leq x \leq e + n \cdot \theta$	PSPACE-complete
$x - y \sim q$	PSPACE-complete	$d + n \cdot \theta \leq x - y \leq e + n \cdot \theta$	PSPACE-complete
$x \sim e$	Undecidable	$x \sim q \cdot y$	Undecidable
$x - y \sim e$	Undecidable	$x - y \sim q \cdot z$	Undecidable

5.2.3. *Parametric TA*

Timing properties of almost all the real-time protocols are typically not concrete but parametric such as message delivery within the time it takes to execute two assignment statements [56]. Concrete timing properties such "as message has to be delivered within 2 time units and an assignment statement has to be executed within 1 time unit" are applicable only for a specific environment. In MDD of real-time systems, parametric timing properties are very appealing for a real-time model of a reusable software module (which is important in MDD of software) or an off-the-shelf real-time hardware (which is gaining popularity in the automotive industry to cope with different *original equipment manufacturers* (OEM) and brands). Moreover, frequently real-time systems are embedded in diverse environments which forces a designer to model the system according to certain parameters. In the early design phase parametric models are usually more convenient for a designer compared to concrete models.

Parametric TA [56], a generalized form of TA, can model parametric timing properties by introducing parametric clock constraints. A parametric TA is a timed automaton which has an accepting run for a parameter valuation of its parametric clock constraints. The emptiness problem for a parametric TA is described as "is there a parameter valuation for which the automaton has an accepting run?".

Theorem 28. *The emptiness checking for parametric TA with three or more clocks is undecidable, while it is decidable with only one clock and is an open problem with two clocks* [56].

Parametric TA can be divided into *linear parametric TA* (where all parametric expressions are linear) and *non-linear parametric TA*. An important subclass of parametric TA is *lower bound automata* [57], in which parameters are only used to calculate the lower bounds in clock constraints. Similarly, the class of *upper bound automata* [57] is a specialization of parametric automata and parameters in upper bound automata are only used to determine the upper bounds in clock constraints. These two classes of automata are together called *lower bound/upper bound automata* or *L/U automata* [57]. Although L/U automata are a restricted form of parametric TA, they can be used to model many noteworthy algorithms and protocols such as *Fisher's mutual exclusion algorithm* [111], and *the root contention protocol* [112].

Theorem 29. *The emptiness checking problem for L/U automata is* PSPACE*-complete* [113,57].

The universality problem for a parametric TA defined as "does a set of parametric valuations contain all the parametric valuations for which the automaton has an accepting run?".

Theorem 30. *The universality checking for L/U automata is* PSPACE*-complete* [113].

Theorem 31. *The model checking problem for L/U automata is also decidable* [113,57].

Model checking for parametric TA is discussed in [114–118]. *TA with ASAP semantics* [52], a variant of parametric TA, is discussed in Section 6.

IMITATOR[11] [119] can extract the largest safe[12] subset of parameter values for a parametric TA from a given set of parameter values. HYTECH[13] [120] is also used for the analysis of parametric TA such as reachability analysis and operations on states set. Using the open source library REDLIB[14] [121], RED[15] [122] also performs parametric safety analysis, simulation checking, and model checking form parametric TA. VerICS[16] [123] and TREX[17] [124] are two other model checkers and analyzers for parametric TA.

5.3. *TA with other clock updates*

Variants of TA which add more expressive clock updates to the existing clock reset of classical TA are discussed in this subsection. Like the variants of Section 5.2, variants of this subsection also fail to retain some important theoretical properties of classical TA.

5.3.1. *Updatable TA*

Theorem 32. *TA with diagonal constraints are exponentially more concise*[18] *than diagonal-free TA* [125].

Theorem 33. *TA with diagonal constraints are not more expressive than diagonal-free TA* [2,5].

Diagonal constraints may yield different behavior in an extension of TA called *updatable TA* [50]. Unlike classical TA, when a switch is taken, an updatable TA can update a specified subset of clocks to values other than 0. An update u of a clock x is deterministic if u has at most one possible value to assign as $\nu'(x)$ for any clock valuation ν, where $\nu'(x)$ is the

[11] For more information on IMITATOR visit http://www.lsv.ens-cachan.fr/~andre/IMITATOR/.

[12] Safe in a sense that the model is guaranteed not to violate a set of specified safety properties.

[13] For more information visit http://embedded.eecs.berkeley.edu/research/hytech/.

[14] http://sourceforge.net/news/?group_id=226122

[15] RED website: http://cc.ee.ntu.edu.tw/farn/red/

[16] URL to know more about VerICS is http://verics.ipipan.waw.pl/

[17] TREX website: http://www.liafa.jussieu.fr/~sighirea/trex/

[18] The *size of an automaton* A, denoted $|A|$, is the length of its (binary) encoding (states and transitions) on the tape of a *Turing Machine*. Automaton A_1 is exponentially more concise than automaton A_2 if these two automata are language equivalent and $|A_1|$ is polynomial in n, where $|A_2|$ is at least exponential in n.

Table 7 – Complexity of reachability checking using different clock updates.

Clock update	Diagonal-free	With diagonal
$x \coloneqq c$	PSPACE-complete	PSPACE-complete
$x \coloneqq y$	PSPACE-complete	PSPACE-complete
$x \coloneqq x+1$	PSPACE-complete	Undecidable
$x \coloneqq y+c$	PSPACE-complete	Undecidable
$x \coloneqq x-1$	Undecidable	Undecidable
$x :< c$	PSPACE-complete	PSPACE-complete
$x :> c$	PSPACE-complete	Undecidable
$x :\sim y+c$	PSPACE-complete	Undecidable
$y+c <: x :< y+d$	PSPACE-complete	Undecidable
$y+c <: x :< z+d$	Undecidable	Undecidable

value of x after the update u. An example of deterministic update is $x \coloneqq c$, whereas $x :> c$ is an nondeterministic update which can assign any value as $\nu'(x)$ which is greater than c.

Theorem 34. *The emptiness checking problem for updatable TA with updates of the form* $x \coloneqq x-1$ *or* $y+c <: x :< z+d$ *is undecidable, where* $c, d \in \mathbb{Q}_+$ [50]. *Only allowing updates of the form* $x \coloneqq c$ *or* $x \coloneqq y$ *or* $x :< c$ *keeps the emptiness checking problem* PSPACE*-complete* [50].

Updatable TA behave surprisingly for the updates of form $x \coloneqq x+1$ or $x \coloneqq y+c$ or $x :> c$ or $x :\sim y+c$ or $y+c <: x :< y+d$; because these updates make the emptiness checking problem for updatable TA with diagonal constraints undecidable, while the emptiness checking problem for diagonal-free updatable TA with these updates is PSPACE-complete [50].

Theorem 35. *Updatable TA for which the emptiness problem is decidable can be converted into equivalent classical TA* [50]. *These decidable updatable TA are more concise than classical TA* [50].

A summary of different kinds of clock updates and their effect on reachability checking for both diagonal-free TA and TA with diagonal clock constraints is shown in Table 7.

5.3.2. *Suspension automata*

A *bounded subtraction clock update* [45] is a clock update of the form $x \coloneqq x-n$ if $n \leq \nu(x) \leq k(x)$, where $n \in \mathbb{N}_0$ and $k(x)$ is the ceiling for x. *Suspension automata* [58], a variant of TA, use *stopwatch*-like clocks and bounded subtraction clock updates along with $x \coloneqq 0$.

Theorem 36. *The language emptiness checking problem and the language inclusion problem for suspension automata are decidable* [58].

However, the language emptiness checking problem for TA with *(unbounded) subtraction clock update* in the form $x \coloneqq x-n$ is undecidable [50].

5.3.3. *Integer reset TA*

The class of *integer reset TA* [126,59] is a subclass of classical TA since it can reset a clock (to zero) only when it has integer value. Switches without reset can occur at any time (including at fractional times). Integer reset TA are less expressive than classical TA, e.g., integer reset TA cannot distinguish between the time stamps of actions occurring within a unit open interval $(i, i+1)$.

Theorem 37. *Although language inclusion for classical TA is undecidable, it is decidable to check whether a timed regular language contains an integer reset timed regular language* [59]. *Contrary to classical TA, integer reset TA are closed under complementation* [59].

5.4. *TA with other clock rates*

The evolving frequency of a clock is called *clock rate* of that clock. All clocks of a classical TA have the same monotone clock rate. Adding different kinds of clock rates with classical TA gives birth to a very expressive, challenging, and popular arena of formal methods called formal methods for hybrid systems. Many researchers consider these variants a completely separate class from TA called *hybrid automata* [61].

5.4.1. *Rectangular automata and controlled TA*

Each clock may have a different clock rate in *rectangular automata* [127,62] which is an interesting extension of TA. Each clock rate is bounded by upper and lower bound constants. In a rectangular TA each clock can have its own and bounded variable clock rate. A clock can change its clock rate only after performing reset operation in *initialized rectangular* TA.[19]

Theorem 38. *For each initialized rectangular automaton there is an equivalent TA* [62].

Thus the reachability problem is decidable for this variant. Relaxing either the clock rate boundedness or the initialization assumption leads to undecidability of the reachability problem. Like rectangular automata, clocks in *controlled TA* [63] have variable clock rates. Controlled TA also allow periodic clock constraints and stopwatch-like clocks. *Stopwatches automata* [64], *interrupt TA* [67], and *distributed time-asynchronous automata* [65] are other two general variants of TA which use stopwatch to increase the expressive power of TA. *Distributed TA with independently evolving clocks* [66] are inspired by distributed time-asynchronous automata and execute in a network of TA each of which may have different clock rates.

Perturbed TA [69] is a special kind of (initialized) rectangular TA which considers *timing perturbation*. *Robust TA* [68] and TA with ASAP semantics [52] are two other well-known variant of TA which considers perturbation. Section 6 presents more discussion on timing perturbation of TA.

5.4.2. *Hybrid automata*

Hybrid systems (e.g., *biological cell networks* [128]) are described by the combination of analog and digital inputs and outputs. Hybrid automata [61], (probably) the most famous and most expressive generalization of TA, can model hybrid systems. Hybrid automata thus model discrete controllers embedded within an analog environment (e.g., a digitally controlled drone flies in a continuously changing environment).

A hybrid automaton is a finite automaton associated with real-valued variables whose trajectories obey general

[19] The initialization property states that whenever the rate of a clock changes it must be reset.

dynamic laws described by differential equations. Under specified conditions a hybrid automaton can change to different dynamic laws. There are many subclasses of hybrid automata which are not TA such as *(non-initialized) rectangular automata* [127], *affine hybrid automata* [128], *polynomial hybrid automata* [129]. The area of hybrid automata is exceedingly large (for example TA can be seen as a subclass of hybrid automata) and out of the scope of this survey. Interested readers can read surveys on hybrid automata [130–135].

5.5. TA with resources

This group of variants has been introduced almost a decade after the introduction of classical TA. This group of variants has quickly received a lot of attention because of the significance of resources in real-time systems. Now there are (at least) 18 variants which can be classified as TA with resources. No other class of TA has so many variants.

5.5.1. Weighted TA or priced TA

In MDD, a timed automaton serves as a superior model for a real-time system over a finite state automaton because TA can explicitly assert time constraints. However, a classical TA is unable to inform the designer how many resources (bandwidth, power, development time, money, etc.) its implementation will consume. This resource consumption information (especially optimal resource consumption) may play a crucial role in MDD. A designer can extract the total resource consumption information of the implementation from a model if the designer attaches a resource consumption function to each state and to each transition of that model. If the resource consumption is proportional to the units of time the implementation stays in a state, then a timed automaton with resource consumption functions is more desirable than a finite state automaton with resource consumption functions. After recognizing the absence of TA with resource consumption functions, Alur et al. and Larsen et al. independently introduced TA with resource consumption functions in 2001, and called them *weighted TA* [43] and *priced TA* [46], respectively.

A *weighted/priced TA* consists of a timed automaton A and a price/cost function $\mathcal{P}$ that maps every location $l \in L$ and every switch $e \in E$ to a nonnegative rational number: $\mathcal{P}(l)$ is the cost for staying in l per unit of time and $\mathcal{P}(e)$ is the cost for performing the switch e. Thus, every run in a weighted/priced automaton has its own accumulated cost and an automaton may have many runs. As a result, to reach a location from a source location with optimal cost (minimum or maximum cost) is an important decision problem for weighted/priced automata. This problem is called *optimal reachability* and is decidable [43,46]. Optimal reachability is a general form of minimum-time reachability.

Theorem 39. *The optimal-reachability problem for weighted/priced TA is* PSPACE*-complete* [136].

Optimal reachability and *optimal scheduling* using weighted/priced automata are well studied and supported in UPPAAL CORA,[20] a variant of UPPAAL, which is a specialized tool for optimal reachability and optimal scheduling [44,137,138]. REMES-IDE can transform REMES[21] [139] (REsource Model for Embedded Systems) models into behaviorally equivalent weighted/priced TA [140]. REMES-IDE provides a graphical editor for the resulting priced automata, as a tool to visually inspect transformation results. Model files for both UPPAAL (TA) and UPPAAL CORA (weighted/priced TA) can be exported to REMS-IDE for verification and analysis.

A timed-cost-extended version of CTL (branching-time temporal logic) is WCTL, while WMTL is a timed-cost-extended version of LTL (linear-time temporal logic). Model checking with respect to WCTL is undecidable for weighted/priced TA with three clocks or more [141]. Model checking of weighted/priced TA with one clock with respect to WCTL is PSPACE-complete [142], while the decidability of WCTL model checking for weighted/priced automata with two clocks is still open [138]. However, model checking with respect to WMTL is decidable only for one clock weighted/priced TA using a single stopwatch-cost[22] variable [143].

Because of the good analytical power of weighted/priced automata, they have been studied extensively and many variants have been proposed such as *uniformly-priced TA* [70], *dual-priced TA* [71], *multi-priced TA* [71], *concavely-priced TA* [74], *priced timed game automata* [77], *concavely-priced probabilistic TA* [75], *weighted integer reset TA* [60], and *priced probabilistic TA* [72,144]. More interesting information about weighted/priced automata may be found in a recent article [145] by Bouyer et al.

5.5.2. Task automata or TA extended with real-time tasks

Finite automata can only describe the *arrival sequence* among the actions, while classical TA can describe both the arrival sequence among the actions and the *arrival time* of an action. Like finite automata, classical TA also describe every action as an instantaneous instance. It is not clear, however, how every action of every real-time system can be instantaneous. This assumption makes classical TA a restrictive or very abstract model for real-time systems. Norström et al. [73] have extended TA by adding real-time tasks with actions. Later on their work evolved into *task automata* or TA extended with real time tasks with locations [45,7,146]. A task is an executable program. A task can be described by its *task type* (or *task name*),[23] *best case computational time*, *worst case computational time*, *relative deadline*,[24] (optionally) *priority for scheduling*, and (occasionally) resource consumption information. Task automata may model a real-time system which is composed of both *periodic tasks* and *sporadic tasks*. Task automata are at least as expressive as classical TA [45,73].

[20] http://www.cs.aau.dk/~behrmann/cora/.

[21] A REMES model is a *state-machine* based behavioral language with support for hierarchical modeling, resource annotations, continuous time, and notions of explicit entry and exit points that make it suitable for component-based modeling of embedded systems. For more information visit http://www.fer.hr/dices/remes-ide.

[22] A cost is stopwatch if it behaves like a clock that can be stopped and restarted.

[23] There can be more than one task with the same name or type.

[24] Relative deadline depends on a tasks release time from the location.

A task automaton is a restricted updatable TA with an extra clock x_{done} and a partial function $\mathcal{F}$. Like suspension automata, task automata only allow bounded subtraction updates along with classical reset to zero. Clock x_{done} is reset whenever a task finishes. The partial function $\mathcal{F}(l)$[25] may annotate a location l with a task. An incoming switch triggers an instance of each annotated task of a location; thus the associated guard of that incoming switch specifies the possible trigger time of the annotated task(s) of a location. A task is entered into an task queue (i.e., the ready queue in an operating system) whenever it is triggered. In an execution queue, tasks are executed according to a given scheduling algorithm, e.g., *fixed priority scheduling* or *earliest deadline first*. In a *fixed task automaton* all the tasks have a *constant computational time*,[26] while the computational time of a task may vary in a *flexible task automaton*. The control behavior of a *feedback task automaton* can be influenced by the reset of a clock x_{done} (or the actual finishing time of a task); on the other hand, the actual finishing time cannot influence the control of a *non-feedback task automaton*. A non-feedback task automaton does not have any clock constraint which includes x_{done}.

In MDD of real-time systems, one of the most significant concerns is *schedulability analysis* prior to implementation. Classical TA do not have any support for specifying resource consumption information and computational time information; although classical weighted/priced TA can specify resource consumption information, they are unable to specify different computational time information such as best case computational time, worst case computational time, etc. A task automaton is schedulable if there exists a scheduling strategy such that all possible sequences of actions generated by the automaton are schedulable in the sense that all associated tasks can be computed within their deadlines. Scheduling algorithms can be divided into *preemptive scheduling algorithms* and *non-preemptive scheduling algorithms*: a currently executed task can be pre-empted by another task in a preemptive scheduling algorithm, whereas, a currently executed task cannot be pre-empted by another task in a non-preemptive scheduling algorithm.

Theorem 40. *The* non-preemptive schedulability checking problem *of a task automaton can be converted into the reachability problem of a classical TA and thus it is* PSPACE*-complete* [45,73].

Theorem 41. *The* preemptive schedulability checking problem *of a task automaton can also be converted into the reachability problem of a classical TA if that automaton is not both a flexible task automaton and a feedback task automaton* [45]. *The preemptive schedulability checking problem of a task automaton is undecidable if it is both a flexible task automaton and a feedback task automaton* [45].

Table 8 shows the schedulability problem is undecidable only for a small class of task automata.

[25] $\mathcal{F}(l)$ is a partial function because some locations may not have any annotated task.

[26] If task has a constant computational time then the best case computational time is equal to the worst case computational time.

Boundedness checking[27] is a useful analysis for a model in MDD of a real-time system, because it can be used for the estimation of the memory consumption by the implementation of that model. If every task has finite size then schedulability implies boundedness but not the other way around, as a bounded ready queue may not be schedulable. As a result, boundedness checking is decidable for a task automaton if that automaton is schedulable [147]. The memory allocated for a bounded task queue can be fixed at compile-time and no exception handling for a queue overflow is needed at run time. A real-time model can exhibit *Zeno* behavior, i.e., infinite sequence of action transitions occur within a finite time unit. Such a model is not implementable and also non-schedulable. It is possible to impose a deterministic semantics of a task automaton and also preserve the safety properties satisfied by the non-deterministic semantics [147]. Non-determinism among action transitions is resolved by assigning unique priorities to the action transitions and non-determinism among time transitions is resolved by implementing the maximal-progress assumption [148]. The expressive power, available analyses (schedulability, boundedness checking, non-Zenoness checking, resource consumption computation) and deterministic semantics make task automata a suitable model supporting for the MDD of real-time systems. In particular, it is a good model for code synthesis if the target platform ensures the *synchrony hypothesis*, i.e., the run-times of related system functions are negligible compared to the different execution times of the associated tasks of the model. TIMES[28] [147], based on task automata, is a popular tool in the research community for real-time code synthesis and scheduling. Schedulability analysis problems of task automata for multi-processor platforms have been studied in [149].

5.5.3. *Timed P automata*

A biologically inspired (specifically, the structure and the functioning of living cells) model called *P systems*[29] [150] has received huge attention[30] in the area of theoretical computer science for its impressive computational and modeling power. Membrane computing naturally models mobility, distributed parallel computing, biomolecular systems, and ecological systems. A P system comprises a hierarchy of membranes; each of these membranes contains a multiset of reactant objects and (possibly) other membranes. An evaluation rule describes reactants and the resulting product. An evaluation rule can be applied only to objects of that membrane.

In *timed P systems* [151], a variant of P systems, each evolution rule is associated with an integer which represents the number of time units needed by the rule to be entirely executed. Recently proposed *timed P automaton* [76] is a timed

[27] The boundedness checking problem is to check whether the size of the task queue for all reachable states is bounded.

[28] For more information visit http://www.timestool.com.

[29] P Systems was introduced in 1998 by Gheorghe Păun, whose last name is the cause of the letter P in 'P Systems'. For more information on P systems please visit http://ppage.psystems.eu/.

[30] On 3rd October 2003, Membrane Computing was selected by Thomson Institute for Scientific Information (ISI) as a "Fast Emerging Research Front in Computer Science".

Table 8 – Complexity of preemptive and non-preemptive scheduling of task automata.

Task automata	Preemptive scheduling	Non-preemptive scheduling
Fixed and feedback	PSPACE-complete	PSPACE-complete
Fixed and non-feedback	PSPACE-complete	PSPACE-complete
Flexible and feedback	Undecidable	PSPACE-complete
Flexible and non-feedback	PSPACE-complete	PSPACE-complete

automaton with a discrete time domain where every location is a timed P system. Timed P automata are useful to study a population which dynamically changes with time (e.g., the population of a place whose dynamics changes with seasons).

5.6. TA with probability

Any real-time property can be either a *hard real-time property* (e.g., "the car stops within 800 time unit after the break is applied") or a *soft real-time property* (e.g., "at most 3% of all the messages will not be delivered within 5 unit of times"). While hard real-time properties are essential in many safety critical real-time systems (e.g., robotic surgery), soft real-time properties are required for many commonly used real-time systems (e.g., video streaming). Unfortunately, classical TA and all its variants discussed above cannot support soft real-time properties. To serve as a complete model for the MDD of real-time systems, TA have to have support for the specification and analysis of soft real-time properties along with hard real-time properties. Soft real-time properties are frequently used in fault tolerant real-time systems (e.g., communication protocols, multimedia protocols) where hard real-time properties are too restrictive: violating a hard deadline does not affect the functionality of the protocol. Soft real-time properties are supported by a popular extension of TA called *discrete probabilistic TA* [78,47,152,48]. Recently, an expressive generalization of discrete probabilistic TA has been proposed called *first-order probabilistic TA* [80]. Hierarchic first-order superposition-based theorem proving and probabilistic model checking both are useful for models based on first-order probabilistic TA-based models.

Clocks in a *continuous probabilistic TA* [79] can be reset according to *continuous probability distributions*. On top of soft deadline properties, continuous probabilistic TA also enable *stochastic timing*, that is, soft deadlines must be satisfied under the assumption that some set of events is influenced by a certain continuous time probability distribution. An example [153] of stochastic timing properties is "the arrival rate of video frames is normal with mean of 40 ms and variance of 5 ms, and service is exponential with rate 45 ms". Thus stochastic timing properties can estimate some important *performance parameters* such as *throughput* and *mean service time*.

Every switch of a probabilistic TA encodes its likelihood to occur. This likelihood is calculated from the execution of certain actions by the system. Hence, probabilistic TA can be used to evaluate *quality of service* which is the quantitative estimation of the probability of achieving some target (e.g., perform a certain task in a time bound). Timed probabilistic properties can be expressed by *probabilistic timed branching-time temporal logic (PTCTL)* [48].

Theorem 42. *Model checking for PTCTL can be performed by converting it into model checking for probabilistic branching-time temporal logic (PTCL)* [48].

Zone-based forward and backwards PTCTL symbolic model checking also has been studied in the literature [48,154]. Among tools, UPPAAL PRO[31] and Fortuna [144] can analyze *maximum probabilistic reachability properties* of probabilistic TA. The latest version of PRISM[32] (PRISM 4.0) [155] provides more general support for the verification and analysis of both discrete and continuous probabilistic TA. mcpta [156] is another model checker for probabilistic TA.

5.7. TA with communication

Concurrent and communicating models are ideal to model mobile systems, cloud computing, and concurrent embedded systems. Untimed concurrent and communicating models widely use FIFO channels (queues) to communicate among them; channels are also common in real-time concurrent and communicating models such as *communicating real-time state machines* [157], π_{klt}*-calculus* [158], and UPPAAL-models [13]. In 2006, Krcál and Yi developed *communicating TA* [81]. A communicating TA is a network of TA extended with (unbounded) channels. Untimed communicating finite state models are not more for expressive than (classical) finite state automata. A communicating TA with only one channel and no sharing states has the power of a one-counter machine. In contrast, a communicating TA with only two channels and no sharing states has the power of two-counter machines (or Turing machines), thus channels make the verification of communicating TA more difficult [13]. Other TA variants which also use channels to communicate are *multi-queue discrete TA* [83], *omega deterministic timed alternating finite automata* [84], synchronized concurrent TA [85], and queue-connected discrete TA [86]. An interesting TA variant with communication is *phase event automata* [87] which combines both state-base (e.g., Kripke structure) and event-based (e.g., finite state automata) structures. The benefit is one can combine the benefits of both process algebra (which depends on event-base structure) and model-checking (which depends on state-base structure). The trade-off for this kind of structure in practice is that the chance of manual errors during modifying is increased.

A state in a *hierarchical state machine* can be either a normal state or a super-state (which contains some other

[31] http://www.cs.aau.dk/~arild/uppaal-probabilistic/.

[32] PRISM is a well established and popular open source verification tool for probabilistic models such as Markov chains, Markov decision processes, probabilistic automata. For more information visit http://www.prismmodelchecker.org/.

states). Although hierarchical state machines (e.g., STATECHARTS [159], UML[33] [160]) are a widespread model in MDD, very little research has been done to understand their theoretical aspects such as expressiveness, decision problems, concurrency complexity, and formal (unambiguous) semantics. Alur et al. [161] published one of the first publications on the topic of the decision problems and succinctness of (untimed) *communicating hierarchical state machine*. Inspired by their work another group came up with *communicating hierarchical TA* [82,162] to study theoretical aspects of real-time hierarchical state machines.

Theorem 43. *Like (untimed) communicating hierarchical state machines, the reachability problem for communicating hierarchical TA is EXPSPACE-Complete.*

Beyer and Rust developed a hierarchical variant of TA for modular specification. The name of their variant is *Cottbus TA* [89] which was developed in Cottbus, Germany. Rabbit[34] [163] is a model checker (reachability and refinement-checker) for Cottbus TA. Another hierarchical TA variant is *timed cooperating automata* [88,164] which is a real-time variant of *cooperating automata* [165].

5.8. TA with determinizability

Alur, Fix, and Henzinger [42] proposed a determinizable subclass of TA named *event-clock automata* after determining that the major obstacle to achieving determinizability of classical TA is nondeterministic clock resets. All the clocks in an event-clock automaton are divided into two disjoint sets: one set contains only *event-recording clocks* and another set has only *event-predicting clocks*. Every action (or event) in event-clock automata has a one-to-one relation with an event-recording clock and with an event-predicting clock. All the clocks in an event-recording automaton are associated with actions and the number of actions are fixed, thus the number of clocks is fixed. An event-recording clock records when the associated action occurred the last time, and an event-predicting clock shows when the associated action will occur next time. Event-clock automata do not have any ϵ-transitions. Removing all the event-predicting clocks from an event-clock automaton will convert it into an *event-recording automaton*. Similarly, eliminating all the event-recording clocks from an event-clock automaton will transform it into an *event-predicting automaton*.

Event-clock (or event-recording or event-predicting) automata are determinizable thus they are closed under complement. Event-clock (or event-recording or event-predicting) TA are closed under all the Boolean operations.

Theorem 44. *The language-inclusion problem for event-clock automata is PSPACE-complete* [42].

Dima defined a class of regular expressions equivalent to event-clock automata [166]. D'Souza discussed the logical characterization of event-clock automata and event-recording automata [167,168]. *Event-clock visibly pushdown automata* [96] and *recursive event-clock automata* [91] have also been proposed for determinizable self-embedded recursive TA. *Product interval TA* [92] are a subclass of event-recording automata that can be used to model the timed behavior of asynchronous digital circuits. Other related TA variants are *TA with input-determined guards* [93], *eventual TA* [90], *counter-free input-determined TA* [95], and *continuous TA with input-determined guards* [94]. TEMPO [169] is a model checker for event-recording automata and was first released in 2001.

5.9. TA with self-embedded recursion

Self-embedded recursion[35] can model naturally the control flow of sequential computation in typical programming languages with nested and recursive invocations of program modules. A *pushdown TA* [49] is a variant of classical TA which can express real-time self-embedded recursive properties by augmenting a timed automaton with a stack. Many real-time non-regular properties are required for real-time software verification. Unfortunately, introducing self-embedded recursion destroys many important closure properties (e.g., intersection) for modeling and verification. Therefore, these properties are usually handled by less expressive but practically efficient *finite indexing* techniques such as bounded real-time model-checking [170].

Theorem 45. *The binary reachability of a pushdown TA is decidable* [49].

The binary reachability of *past pushdown TA* [97], a parametric variant of *discrete pushdown TA* where the past-formulas[36] can be used as clock constraints, is also decidable.

Theorem 46. *The universality problem and language inclusion problem for* timed visibly pushdown automata [98] *(nondeterministic timed version of* visibly pushdown automata [171]*) even with a single clock is undecidable.*

A deterministic TA version of visibly pushdown automata called event-clock visibly pushdown automata [96] is closed under Boolean operations. It is decidable to check whether a timed visibly pushdown language is included in an event-clock visibly pushdown language [96]. In 2010, *recursive TA* [99] and *timed recursive state machines* [100] were proposed.

5.10. TA with succinctness

The main motivation behind the creation of this group of TA variants is to improve modeling rather than to achieve better analyses.

5.10.1. Alternating TA

An (untimed) *alternating finite automaton* [172] is a nondeterministic finite automaton whose transitions are divided into existential and universal transitions. For example, let A be an alternating automaton. For an existential transition

[33] For more information on UML, please visit http://www.omg.org/spec/UML/2.1.2/.

[34] Rabbits website: http://www.sosy-lab.org/~dbeyer/Rabbit/.

[35] Balanced parentheses languages are well known examples for self-embedded recursion.

[36] A past formula is a formula which includes the past parametric values.

$(s_1, a, s_2 \vee s_3)$, A nondeterministically chooses to switch the state to either s_2 or s_3 after reading a (like a nondeterministic finite automaton). For a universal transition $(s_1, a, s_2 \wedge s_3)$, A moves to s_2 and s_3 after reading a (which simulates the behavior of a parallel machine). An alternating finite automaton accepts a word if there exists a run tree on that word such that every path ends in an accepting state. Due to the universal quantification, a run is represented by a run tree. Any alternating finite automaton is equivalent to a nondeterministic finite automaton. Alternating models are useful to express clauses which are combined by Booleans.

Alternating (tree) TA [173], a real-time extension of alternating automata, are closed under all Boolean operations [105,174].

Theorem 47. *Emptiness checking for alternating TA is decidable only for one clock over finite timed words; any extension (infinite timed words, more than one clock, ϵ-transitions) leads to undecidability* [105,174].

Undecidability proofs of the emptiness checking problem for alternating TA with one clock over infinite words rely on the ability to express "infinitely often" properties. *Weak alternating TA* [106] do not permit one to express "infinitely often" properties, thus the emptiness checking problem for weak alternating TA over infinite words is decidable. Interestingly, bounded time model checking of alternating TA (over finite or infinite words) is decidable as in bounded time the emptiness checking is decidable [175]. TCTL model checking for alternating TA has also been discussed [173].

5.10.2. *TA with deadlines*

Urgency (*urgent transitions* and *urgent locations*) is a common and important concept in real-time models (such as in timed Petri nets [176] or in timed I/O automata [177]) because they allow more succinct representation and resolution of nondeterminism in real-time concurrent models. When an urgent transition (switch) is enabled the control of the TA has to perform the transition instantaneously without spending any time at that location. All the transitions originating from an urgent location are urgent transitions. To our knowledge, urgency has been first introduced by Bornot et al. with TA as *TA with deadlines* [101]. Later on many others generalized TA with deadlines. Among them Barbuti and Tesei proposed a model which is called *TA with urgent transitions* [104]. In Barbuti's model, an urgent transition must be performed within a fixed time interval from its enabling time and a urgent transition has higher priority than other non-urgent transitions enabled in the same state. Although from a language point of view TA with urgent transitions are not more expressive than classical TA, from a specification point of view the use of urgent transitions allows shorter and clearer specifications of urgent and periodic behaviors. *Variable-driven timed automata* [103] and *prioritized TA* [102] are two additional TA variants which mainly focus on urgency issues. UPPAAL [13], UPPAAL TIGA [178] and many other tools use urgency for the specification of their models.

5.11. *TA with games*

A classical TA models only closed real-time systems (where every thing is controlled) while there exist many open real-time systems which interact with uncontrolled environments (or other systems) and these uncontrolled environments influence the behavior of those systems. A good example of real-time open systems is a pacemaker (an open system) which continuously interacts with a heart (an uncontrolled environment). Pacemaker's performance crucially depends on the exact timing of an action performed either by the system or by the environment. *Timed game automata* [107] along with their controller synthesis strategies have been introduced to model such open real-time systems. The *game reachability* problem is whether the system has a strategy to reach a target state regardless of how the environment behaves. The *game minimum-time reachability* problem in timed game automata is finding the minimum time required by the system to reach a target state regardless of how the environment behaves.

Theorem 48. *Both the game reachability and the game minimum-time reachability problems for timed game automata are EXPTIME-complete* [179–181].

Bouyer et al. have discussed optimal strategies in priced timed game automata [77] which is a combination of timed game automata and priced TA.

UPPAAL TIGA [178] is a well-known tool for solving games based on timed game automata with respect to reachability and safety properties. Synthia [182], a recently developed tool in Saarland University, Germany, performs verification and controller synthesis for timed game automata.

Variants of this class (TA with games) are a direct generalization of classical TA. This class itself is emerging as a big research area mainly because of its controller synthesis application. As for the case of hybrid automata, this survey does not elaborately discuss the TA with games class because of its vastness. An interested reader of timed games may find the article [183] to be an easy and comprehensive reading.

6. Implementation

Formal verification, control theory, and other model-based analyses lose a lot of their value when the implementation of the model is not accurate. An accurate implementation of a model conveys and contains all the verified, controlled, and analyzed properties of its archetypical model. Moreover, the correspondence between the model and the accurate implementation is precisely understood.

6.1. *Challenges*

The semantic mismatch between the continuous-time of TA and the discrete-time of implementation platforms (e.g., operating system and related hardware are the implementation platform for software) makes the implementation of TA a hard problem. The notion of instantaneous action (which is performed in precisely zero time units) is another barrier to implement TA accurately as in practice no action can be executed in precisely zero time units. This semantic mismatch and practically non-instantaneous actions create (usually tiny amounts of) clock drift and violations of the guards. Furthermore, a timed automaton with Zeno behavior or a behavior where actions have to be executed in less and less time (even

without causing Zeno behavior) cannot be implementable on a finite-speed platform [184].

MDD advocates generating code automatically from models. Automatic accurate code generation from TA has the potential to improve reliability and reduce costs:

i. Automatic accurate code generation from a formal model should be an essential part of safety-critical software development because code synthesis is (probably) the best way to avoid error-prone manual programming. The group of safety-critical software includes safety-critical software for nuclear plants, life-critical software for medical devices, control software for space shuttles, etc. Along with logical time, concrete time is a major concern for almost all safety-critical software. Therefore, accurate code synthesis for TA has vital significance for safety-critical software development as TA are a prominent real-time formal model.
ii. The wages of competent programmers to write accurate real-time code is high. Moreover, accurate manual programming takes a huge amount of time compared to automatic code generation.

6.2. Solutions

Puri [11] has shown that TA are not *robustly safe*.[37] Puri used the *progress cycle assumption*[38] for a region-based search of the strongly connected components to compute the robust reachable set of states of a timed automaton with closed clock constraints. He showed that in a non-robustly safe TA an infinitesimally small amount of timing perturbation can make bad or unsafe states reachable which are unreachable with no perturbation.

Wulf et al. has proposed a platform-dependent parametric semantics of TA called *almost ASAP* [52] to tackle this implementation problem. Instead of instantaneous actions as in classical semantics, in the almost ASAP semantics actions have to be performed within a parametric timing tolerance. Depending on this timing tolerance the clock constraints are enlarged. This parametric timing tolerance changes according to the speed of the implementation platform. Even though the almost ASAP semantics based approach is an ad hoc approach, if a parametric timing tolerance is good for a platform then that parametric time unit is also good for any faster platform. They have also shown how to determine a suitable parametric timing tolerance for a platform by using an algorithm which is inspired by Puri's robust reachable state set finding algorithm. Both these algorithms are based on region graphs thus they are not practically efficient. Zone-based pragmatically efficient algorithms have been offered in different papers [185,186] including one [187] by Dima where he proposed a zone-based algorithm which can also be applied to TA which have open clock constraints. A discussion on the decision problems, formal verifications, and controller syntheses of different kinds of robust TA exists in the literature [69,188,68,189–192].

Another group of researchers has advocated a modeling-based solution [193] to ensure implementability of TA with classical semantics. Their approach utilizes a network of TA to model the behavior of the platform. Their approach is practically inefficient because of the enormous state-space created by that network of TA. Unlike methods based on almost ASAP semantics, this modeling-based approach is not guaranteed to also ensure correctness on a faster platform, i.e., on a faster platform a timed automaton (which is correct on a slower platform) may perform incorrectly by producing more behavior or higher sampling rates.

An alternative way to check implementability of TA is to check its *samplability*. A TA is samplable if its semantics is preserved under a discretization (sampling rate) of time. Sampled semantics of a timed automaton is a finite approximation of its classical semantics. The *sampling problem* of a timed automaton is to decide whether there is a sampling rate such that any untimed behaviors accepted by it with classical semantics can be also accepted by it with sampled semantics. Recently a group from Uppsala University has shown the sampling problem for TA is decidable [194]. Bouyer et al. have addressed both the robustness and samplability of TA [195]. They have shown that any TA can be converted into a new TA with same behavior such that this new TA is both robust and samplable. Sampling rates of TA have been discussed by a few other groups [196,184,197].

Massive industrial demands for accurate code synthesis from high level models (especially real-time models) are currently attracting attention from researchers both from industry and academia. Accurate code synthesis from high level models is (still) challenging because of the wide gap between the high level model and the low level code. One of the earliest successful code synthesis for TA (including its variants) is the code synthesis for task automata which is discussed in Section 5.5.2. Wulf et al. [52] synthesized code for a *Philips audio control protocol* [198] on the LEGO MINDSTORMS platform using their proposed almost ASAP semantics. Another group used *real-time specification for Java (RTSJ)* [199] to generate Java code from TA [200]. They developed a prototype tool called TART[39] [201] which implements their method. Unfortunately, their technique does not perform code validation and their synthesized code does not preserve timing properties of its archetype TA because of the synchrony hypothesis.

In 2010, a group of researchers from the University of Pennsylvania proposed an iterative process to generate accurate code for TA [202]. Instead of code synthesis, their main focus was to ensure that the implementation has the exact same timing properties as its archetypical model. This iterative process consists of a cycle of modeling, model checking, code generation, testing, and reverse engineering. In their approach, reverse engineering and testing steps use empirical (rather than formal) methods and therefore their technique cannot ensure total safety. In the same year, another group independently proposed a similar approach [203].

[37] Robust reachability computes the set of reachable states if there exist timing perturbations up to a certain amount. In a robustly safe TA no unsafe state is robustly reachable.

[38] In the progress cycle assumption it is assumed that each clock is reset at least once in every cycle of a timed automaton.

[39] To download TART please visit http://itee.uq.edu.au/~niusha/TART/.

7. Tools

The rich and strong theoretical foundation of TA makes them a good candidate to use as the underlying formal model for real-time MDD. Region-based approaches are not suitable for practical purposes because of the exponential size of region automata. Most of the tools use zone-based approaches as these approaches are practically more efficient and scalable. Section 3.2.2 describes how zone-based techniques have changed a lot during the last two decades to overcome many deficiencies. This large number of changes has made it challenging to keep these tools up to date. Merely a few number of tools such as UPPAAL [13], RED [122], and VerICS [123] have been actively maintained and have evolved for a long period of time. As zone-based techniques are now well-established, there is a very bright future waiting for TA-based tools. A complete and detailed survey paper on TA-based tools could be an appealing work in the research community. A survey on TA-based tools is a small part of this paper and it provides only a skeleton view of TA-based tool research. Nonetheless, this paper lists (around) forty TA-based tools. We did not personally perform any experiments on these tools. Rather, our survey mostly relied on related publications, websites, and developers of each tool to collect information.

Tables 9 and 10 list some of the TA-based or closely related tools available on the web and in the literature. All these listed tools were developed and are maintained mainly for research and academic purposes. The first column of these tables displays the names of the listed tools; the second column presents the description of the functionality of these tools; at the end, the third and fourth columns show the first release year and the latest release year of these tools. All these release years have been confirmed by the respective developers other than TASM [204], TEMPO [169], HyTech [120], RED [122], TART [201], Fortuna [144], PRISM 4:0 [155], XAL [205], and McAiT [206].

UPPAAL [13] is a very (or the most) successful and influential TA-based (formal analysis and verification) tool. UPPAAL is now also available for commercial use. This tool was developed and is maintained by the UPPAAL research group. UPPAAL group, a TA focused research group, is formally a collaboration between two research groups of Uppsala University, Sweden and Aalborg University, Denmark. UPPAAL PRO [207], UPPAAL PORT [208], UPPAAL CoVer [209], UPPAAL TIGA [178], UPPAAL CORA [44], UPPAAL TRON [210], TIMES [147], CATS [211], SAVE IDE [212], McAiT [206], and TASM [204] are some other TA-based tools that were developed and are maintained by the UPPAAL research group. Tools such as McAiT [206], SAVE IDE [212], TASM [204] have been put into the UPPAAL group because in addition to having been strongly influenced by the UPPAAL tool, some of the major developers (e.g., Wang Yi, Paul Pettersson) of these tools have strong past or present ties with the UPPAAL group. UPPAAL TIGA [178], TIMES [147], and UPPAAL TRON [210] are popular and highly-cited TA-based research tools for controller synthesis, code synthesis, and black-box testing, respectively.

Verimag, France is a leading research center in embedded systems that was (officially) established in 1993. Until 2006, Verimag was active in TA-based tool research and development. Many TA-based tools such as Kronos [14], SynthKro [213], Open-Kronos [20], TAXYS [214], minim [9], RTSpin [215], IF [216], and TReX [124] are developed and maintained by this research center. Kronos [14] is probably the most well-known TA-based Verimag (formal analysis and verification) tool. However, there has been no official release of Kronos in the last 10 years. Laboratory Specification and Verification (LSV), ENS Cachan, France developed a TA-based compositional model checking tool named CMC [217] in 1994 and there is no new version of this tool after 2004. Instead of updating CMC [217], LSV is actively developing a new TA-based tool (IMITATOR [119]) for parametric analysis. Recently Saarland University, Germany has developed two TA-based tools: mcpta [156] for model checking and Synthia [182] for (verification and) controller synthesis. No other group has developed more than one tool of Tables 9 and 10. Almost all the TA-based research tools are developed in Europe. UPPAAL and Verimag are the main two driving forces of TA-based tool research. Release dates in Tables 9 and 10 indicate Verimag has not been very active in this research arena recently. A large number of successful tools, the diversity in tools functionality, and the long maintenance period suggest that the UPPAAL group is the most established group in this arena.

From the tool descriptions of Tables 9 and 10, we can categorize these forty tools into eleven classes depending on their major functionality: *black-box testing and related*, *code synthesis and scheduling*, *controller synthesis*, *component-based development*, *model minimization*, *mixed reality language development*, *web related development*, *parametric analysis and verification*, *probabilistic analysis and verification*, *resource analysis and verification*, and *other analyses and verification*. Table 11 presents all these TA-based (or related) tools and their classification. Table 11 clearly indicate that the majority of TA-based tools is used for real-time analysis and verification purposes and that tools are also beginning to be used for other purposes. The UPPAAL group, Verimag, LSV, and Saarland University usually develop separate tools for each major functionality. On the other hand, developers of RED [122] and VerICS [123] add major functionality (such as parametric analysis, higher level model analysis) with these tools in each new (version) release. It would be a worthy literature to do a survey which would present detailed comparisons (depending on functionality, performance) of all the TA-based (including commercial) tools of the same class.

8. Conclusion

We present a survey on semantics, decision problems, variants, implementability, and tools of timed automata. Section 3.2 is interesting as it informs how symbolic semantics (region and zone) based analysis has evolved with time to make TA more suitable for practical uses such as tool development. There are many data structures (such as *CDD* [228], *CDR* [122], *NDD* [229]) other than DBM [15,16,10] for symbolic semantics of TA; however, our survey did not explore these data structures. We have listed major linguistic properties and decision problems of classical TA in Section 4. Decidability of emptiness checking and undecidability of inclusion problem are the two major results of this section. When we

Table 9 – TA-based or related tools: part 1.

Name	Description	First rele.	Latest rele.
UPPAAL [13]	An integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of TA, extended with data types.	1995	2011
UPPAAL PRO [207]	A tool for probabilistic reachability analysis for probabilistic TA.	2008	2009
UPPAAL PORT [208]	A tool for component-based modeling, simulation, and verification of real-time and embedded systems modeled as TA.	2006	2008
UPPAAL CoVer [209]	A tool for creating test suites from TA with coverage specified by coverage observers.	2005	2009
UPPAAL TIGA [178]	A tool for solving games based on timed game automata with respect to reachability and safety properties.	2005	2011
UPPAAL CORA [44]	A tool for cost optimal reachability analyses for priced TA.	2002	2006
UPPAAL TRON [210]	A black-box conformance testing tool, based on TA, for embedded real-time software.	2004	2009
TIMES [147]	A tool set for modeling, schedulability analysis, synthesis of (optimal) schedules and executable code.	2002	2005
CATS [211]	A tool for compositional timing and performance analysis of real-time systems modeled using TA and the real-time calculus.	2007	2007
SAVE IDE [212]	A tool for design, analysis and implementation of component-based embedded real-time systems using TA.	2008	2009
McAiT [206]	A timing analyzer for multicore real-time software using TA.	2010	2010
TASM [204]	A tool for specification, simulation, and verification of real-time systems using TA.	2007	2008
Kronos [14]	A tool for checking whether a timed automaton satisfies a TCTL formula.	1992	2002
SynthKro [213]	A tool for controller synthesis of TA.	2002	2002
Open-Kronos [20]	A model-checker for timed Büchi automata.	1997	2005
TAXYS [214]	A TA-based tool for the development and verification of real-time embedded systems.	2000	2001
minim [9]	A tool for minimization of TA with respect to time-abstracting bisimulation.	1996	2001
RTSpin [215]	A verification tool which extends Spin with quantitative dense time features using timed Büchi automata.	1993	2004
IF [216]	A validation platform which uses a specification language based on TA extended with discrete data variables, various communication primitives, dynamic process creation and destruction. This language is expressive enough to represent major modeling and programming languages for distributed systems such as real-time SDL and UML.	1998	2004
TReX [124]	A tool for reachability analysis of complex systems modeled as parametric TA.	2000	2006
IMITATOR [119]	A tool for extracting the largest safe subset of parameter values for a parametric TA from a given set of parameter values.	2009	2011
CMC [217]	A tool for compositional model-checking of real-time systems.	1995	2004
Synthia [182]	A tool for verification and controller synthesis for TA.	2011	2011
mcpta [156]	A model checker for probabilistic TA.	2009	2011
MCTA [218]	A model checker for real-time specifications modeled as TA. It can find shortest error trace.	2008	2009
Rabbit [163]	A model checker for Cottbus TA.	1999	2002
MIRELA Framework [219]	A framework which uses mixed reality language MIRELA. MIRELA's compiler generates TA for simulation and verification of time constraints in this framework.	2007	2008
XAL [205]	A web oriented programming language based on TA.	2008	2008
WST [220]	A tool for design, validation and verification of composite Web Services with timed restrictions using TA.	2007	2011
VerICS [123]	A (bounded, unbounded, parametric, and non-parametric) model checker for timed and multi-agent systems modeled by networks of communicating automata such as TA, time Petri nets. It supports model checking for model specified in high level languages such as Promela, UML, Java, and Estelle.	2003	2010

started working in this area we first thought that TA would have at most thirty variants and then after performing this extensive survey we came to realize that TA had many more variants. We have listed around eighty variants in Section 5. We believe there are more variants of TA that exist and that the number is increasing with time. To our knowledge no survey on TA exists which lists at least twenty variants. We also classify these variants into eleven classes depending on their construction and functionality. This classification will help a reader to understand similarities and differences among TA

Table 10 – TA-based or related tools: part 2.

Name	Description	First rele.	Latest rele.
PRISM 4:0 [155]	A verification tool for probabilistic models including probabilistic TA.	2010	2011
AITARTOS [203]	A tool for automatic implementation of TA model in a real-time operating system.	2010	2011
Fortuna [144]	A model checker priced probabilistic TA.	2010	2010
Priced-Timed Maude [221]	An analyzer for priced TA.	2008	2008
RED [122]	A model checker and simulation checker for TA and parametric analyzer for parametric TA.	2000	2011
HyTech [120]	A model checking and analyses tool for linear hybrid automata including parametric TA.	1995	2003
TEMPO [169]	A model-checker for event recording automata.	2001	2001
DREAM [222]	A distributed real-time embedded systems analyzer based on TA.	2005	2007
TART [201]	A prototype tool to generate Java code from TA using RTSJ.	2010	2010
VInTiMe [223]	VInTiMe is a suite of TA-based tools (Lapsus [224], VTS [225], ObsSlice [226], and Zeus [227]) that combines high-level expressive power, unassisted property-preserving model-reduction and low-level distributed model checking power to describe and verify complex real-time systems.	2003	2009

Table 11 – Major purposes of TA-based or related tools.

Purposes	Tools
Black-box testing and related	UPPAAL TRON [210], UPPAAL CoVer [209]
Code synthesis and scheduling	TIMES [147], SAVE IDE [212], AITARTOS [203], TART [201]
Controller synthesis	UPPAAL TIGA [178] [178], SynthKro [213], Synthia [182]
Component-based development	UPPAAL PORT [208], SAVE IDE [212]
Model minimization	minim [9], VInTiMe [223]
Mixed reality language development	MIRELA Framework [219]
Web related development	XAL [205], WST [220]
Parametric analysis and verification	TReX [124], IMITATOR [119], VerICS, HyTech [120], RED [122]
Probabilistic analysis and verification	UPPAAL PRO [207], mcpta [156], PRISM 4:0 [155], Fortuna [144]
Resource analysis and verification	UPPAAL CORA [44], TIMES [147], CATS [211], Fortuna [144], Priced-Timed Maude [221]
Other analyses and verification	UPPAAL [13], TASM [204], McAiT [206], Kronos [14], Open-Kronos [20], TAXYS [214], RTSpin [215], IF [216], CMC [217], MCTA [218], Rabbit [163], VerICS, RED [122], TEMPO [169], DREAM [222], VInTiMe [223]

variants. This paper discusses only major variants of each class. We hope these discussions on major variants will give a reader a rough idea of other variants of the same class. An interested reader can learn more about an undescribed (but listed) variant from its related citation. Use of TA in the industry will be widespread if researchers can triumph over TA's state-space explosion problem and TA's realizability problem. Section 3.2 briefly discusses symbolic semantics which is only one of the frontiers in the war on state-space explosion. Section 6 introduces a reader to the realizability and property preservation problem of TA. Accurate TA implementability is getting more attention every day. Usually robustness analysis introduces larger state-space (e.g., Figure 3 of [190]). A study on the comparison and relation between these two problems (state-space explosion and robust analysis) of TA would be an interesting work for the research community. Section 7 lists, describes, and classifies forty TA-based research and academic tools. A rigorous survey on TA-based tools including case studies, performance analyses, and commercial tools would be a good step to convince people to increase the use of TA in industry.

In only two decades the theory of timed automata has established itself as an integral part of real-time systems development. This area is becoming more and more active. Our survey is only a snapshot of this area. It is impossible to cover the whole area in a single article. The main motivation of this survey was to provide a sorted picture of this scattered arena. This survey did not discuss real-time temporal logics, real-time formal verification, and real-time controller synthesis because these topics are mostly related to real-time formal models in general instead of being specific to timed automata.

Acknowledgments

We want to thank Wang Yi, Patricia Bouyer, Hubert Comon, Catalin Dima, Stavros Tripakis, Sergio Yovine, Paul Pettersson, Holger Hermanns, Alexandre David, Jochen Hoenicke, Didier Lime, Andreas Podelski, Dirk Beyer, Fernando P. Schapachnik, Peter Csaba Ölveczky, Mihaela Sighireanu, Anders Hessel, Marius Mikucionis, Maciej Szreter, Maria Emilia Cambronero

Piqueras, Francois Laroussinie, Gregorio Diaz, Marius Bozga, Bachir Djafri, Enrique Martínez López, Pavel Kučera, Hans-Jörg Peter, Séverine Sentilles, Martin Wehrle, and Franck Cassez for their help regarding their works. We also want to thank James R. Cordy, Ahmed E. Hassan, and Stefan D. Bruda, and anonymous reviewers for their comments and reviews on this survey.

REFERENCES

[1] R. Alur, D.L. Dill, Automata for modeling real-time systems, in: Proceedings of the 17th International Colloquium on Automata, Languages and Programming, Springer-Verlag New York, Inc, New York, NY, USA, 1990, pp. 322–335.
[2] R. Alur, D.L. Dill, A theory of timed automata, Theoretical Computer Science 126 (1994) 183–235.
[3] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine, Symbolic model checking for real-time systems, Information and Computation 111 (1994) 394–406.
[4] R. Alur, Timed automata, Theoretical Computer Science 126 (1999) 183–235.
[5] B. Bérard, A. Petit, V. Diekert, P. Gastin, Characterization of the expressive power of silent transitions in timed automata, Fundamenta Informaticae 36 (1998) 145–182.
[6] R. Alur, C. Courcoubetis, D. Dill, Model-checking in dense real-time, Information and Computation 104 (1993) 2–34.
[7] E. Fersman, P. Pettersson, W. Yi, Timed automata with asynchronous processes: Schedulability and decidability, in: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'02, Springer-Verlag, London, UK, 2002, pp. 67–82.
[8] R. Alur, C. Courcoubetis, N. Halbwachs, D.L. Dill, H. Wong-Toi, Minimization of timed transition systems, in: Proceedings of the 3rd International Conference on Concurrency Theory, CONCUR'92, Springer-Verlag, London, UK, 1992, pp. 340–354.
[9] S. Tripakis, S. Yovine, Analysis of timed systems using time-abstracting bisimulations, Formal Methods in System Design 18 (2001) 25–68.
[10] D.L. Dill, Timing assumptions and verification of finite-state concurrent systems, in: Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems, Springer-Verlag New York, Inc, New York, NY, USA, 1990, pp. 197–212.
[11] A. Puri, Dynamical properties of timed automata, in: Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'98, Springer-Verlag, London, UK, 1998, pp. 210–227.
[12] M. Sorea, Verification of real-time systems through lazy approximations, Ph.D. Thesis, Universität Ulm, Germany, 2003.
[13] G. Behrmann, A. David, K.G. Larsen, P. Pettersson, W. Yi, Developing UPPAAL over 15 years, Software: Practice and Experience 41 (2) (2011) 133–142. ISSN 0038-0644.
[14] C. Daws, A. Olivero, S. Tripakis, S. Yovine, The tool KRONOS, in: Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control, Springer-Verlag New York, Inc, Secaucus, NJ, USA, 1996, pp. 208–219.
[15] R. Bellman, Dynamic Programming, Princeton University Press, 1957.
[16] J. Bengtsson, Clocks, DBMs and states in timed systems. Ph.D. Thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2002.
[17] J. Bengtsson, W. Yi, Timed automata: semantics, algorithms and tools, in: W. Reisig, G. Rozenberg (Eds.), Lecture Notes on Concurrency and Petri Nets, in: Lecture Notes in Computer Science, vol. 3098, Springer–Verlag, 2004.
[18] C. Daws, S. Tripakis, Model checking of real-time reachability properties using abstractions, in: Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems, Springer-Verlag, London, UK, 1998, pp. 313–329.
[19] A. Bouajjani, S. Tripakis, S. Yovine, On-the-fly symbolic model checking for real-time systems, in: Proceedings of the 18th IEEE Real-Time Systems Symposium, RTSS'97, IEEE Computer Society, Washington, DC, USA, 1997, pp. 232–243.
[20] S. Tripakis, S. Yovine, A. Bouajjani, Checking timed Büchi automata emptiness efficiently, Formal Methods in System Design 26 (2005) 267–292.
[21] S. Tripakis, Checking timed Büchi automata emptiness on simulation graphs, ACM Transactions on Computational Logic 10 (2009) 15: 1–15: 19.
[22] T.G. Rokicki, Representing and modeling digital circuits, Ph.D. Thesis, Department of Computer Science, Stanford University, Stanford, CA, USA, 1993.
[23] J. Bengtsson, W. Yi, On clock difference constraints and termination in reachability analysis of timed automata, in: J.S. Dong, J. Woodcock (Eds.), Proceedings of the 5th International Conference on Formal Engineering Methods, in: Lecture Notes in Computer Science, vol. 2885, Springer–Verlag, 2003.
[24] P. Bouyer, Forward analysis of updatable timed automata, Formal Methods in System Design 24 (2004) 281–320. ISSN 0925-9856.
[25] P. Bouyer, F. Laroussinie, P.-A. Reynier, Diagonal constraints in timed automata: forward analysis of timed systems, in: P. Pettersson, W. Yi (Eds.), Proceedings of the 3rd International Conference on Formal Modelling and Analysis of Timed Systems, FORMATS'05, in: Lecture Notes in Computer Science, vol. 3829, Springer, Uppsala, Sweden, 2005, pp. 112–126.
[26] P.-A. Reynier, Diagonal constraints handled efficiently in UPPAAL. Technical Report LSV-07-02, Laboratoire Spcification et Vrification, ENS Cachan, France, 2007.
[27] E. Asarin, P. Caspi, O. Maler, Timed regular expressions, Journal of the ACM 49 (2) (2002) 172–206.
[28] P. Bouyer, A. Petit, A Kleene/Büchi-like theorem for clock languages, Journal of Automata, Languages and Combinatorics 7 (2) (2002) 167–186.
[29] C. Dima, Regular expressions with timed dominoes, in: Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science, DMTCS'03, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 141–154.
[30] C. Dima, Timed Shuffle Expressions, Springer-Verlag, London, UK, 2005, pp. 95–109.
[31] O. Finkel, On the shuffle of timed regular languages, Bulletin of the European Association for Theoretical Computer Science 88 (2006) 182–184.
[32] R. Alur, P. Madhusudan, Decision problems for timed automata: a survey, in: M. Bernardo, F. Corradini (Eds.), Formal Methods for the Design of Real-Time Systems — Revised Lectures of the International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT'04, in: Lecture Notes in Computer Science, vol. 3185, Springer-Verlag, 2004, pp. 1–24.
[33] C. Courcoubetis, M. Yannakakis, Minimum and maximum delay problems in real-time systems, Formal Methods in System Design 1 (1992) 385–415.
[34] P. Niebert, S. Tripakis, S. Yovine, Minimum-time reachability for timed automata, in: Proceedings of the 8th Mediteranean Conference on Control and Automation, MED2000.
[35] R. Alur, C. Courcoubetis, T.A. Henzinger, The observational power of clocks, in: Lecture Notes in Computer Science, vol. 836, Springer-Verlag, 1994, pp. 162–177.

[36] K. Cerans, Decidability of bisimulation equivalences for parallel timer processes, in: Proceedings of the 4th International Workshop on Computer Aided Verification, Springer-Verlag, London, UK, 1993, pp. 302–315.
[37] K.G. Larsen, Y. Wang, Time-abstracted bisimulation: implicit specifications and decidability, Information and Computation 134 (1997) 75–101.
[38] S. Tasiran, R. Alur, R.P. Kurshan, R.K. Brayton, Verifying abstractions of timed systems, in: Proceedings of the 7th International Conference on Concurrency Theory, CONCUR'96, Springer-Verlag, London, UK, 1996, pp. 546–562.
[39] O. Finkel, Undecidable problems about timed automata, in: Proceedings of the 4th international conference on Formal Modeling and Analysis of Timed Systems, FORMATS'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 187–199.
[40] S. Tripakis, Folk theorems on the determinization and minimization of timed automata, Information Processing Letters 99 (6) (2006) 222–226.
[41] H. Comon, Y. Jurski, Timed automata and the theory of real numbers, in: Proceedings of the 10th International Conference on Concurrency Theory, CONCUR'99, Springer-Verlag, London, UK, 1999, pp. 242–257.
[42] R. Alur, L. Fix, T.A. Henzinger, Event-clock automata: a determinizable class of timed automata, Theoretical Computer Science 211 (1999) 253–273.
[43] R. Alur, S.L. Torre, G.J. Pappas, Optimal paths in weighted timed automata, in: Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control, HSCC'01, Springer-Verlag, London, UK, 2001, pp. 49–62.
[44] G. Behrmann, K.G. Larsen, J.I. Rasmussen, Optimal scheduling using priced timed automata, ACM SIGMETRICS — Performance Evaluation Review 32 (2005) 34–40.
[45] E. Fersman, P. Krčál, P. Pettersson, W. Yi, Task automata: schedulability, decidability and undecidability, International Journal of Information and Computation 205 (2007) 1149–1172.
[46] G. Behrmann, A. Fehnker, T. Hune, K.G. Larsen, P. Pettersson, J. Romijn, F.W. Vaandrager, Minimum-cost reachability for priced timed automata, in: Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control, HSCC'01, Springer-Verlag, London, UK, 2001, pp. 147–161.
[47] D. Beauquier, On probabilistic timed automata, Theoretical Computer Science 292 (2003) 65–84.
[48] M. Kwiatkowska, G. Norman, R. Segala, J. Sproston, Automatic verification of real-time systems with discrete probability distributions, Theoretical Computer Science 282 (2002) 101–150.
[49] Z. Dang, Pushdown timed automata: a binary reachability characterization and safety verification, Theoretical Computer Science 302 (2003) 93–121.
[50] P. Bouyer, C. Dufourd, E. Fleury, A. Petit, Updatable timed automata, Theoretical Computer Science 321 (2004) 291–345.
[51] C. Baier, N. Bertrand, P. Bouyer, T. Brihaye, M. Größer, Probabilistic and topological semantics for timed automata, in: Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 179–191.
[52] M. De Wulf, From Timed Models to Timed Implementations. Thèse de doctorat, Département d'Informatique, Université Libre de Bruxelles, Belgium, December 2006.
[53] C. Choffrut, M. Goldwurm, Timed automata with periodic clock constraints, Journal of Automata, Languages and Combinatorics 5 (2000) 371–403.
[54] B. Bérard, C. Dufourd, Timed automata and additive clock constraints, Information Processing Letters 75 (2000) 1–7.
[55] J.S. Miller, Decidability and complexity results for timed automata and semi-linear hybrid automata, in: Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control, HSCC'00, Springer-Verlag, London, UK, 2000, pp. 296–309.
[56] R. Alur, T.A. Henzinger, M.Y. Vardi, Parametric real-time reasoning, in: Proceedings of the 25th Annual ACM Symposium on Theory of Computing, STOC'93, ACM, New York, NY, USA, 1993, pp. 592–601.
[57] T. Hune, J. Romijn, M. Stoelinga, F.W. Vaandrager, Linear parametric model checking of timed automata, in: Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001, Springer-Verlag, London, UK, 2001, pp. 189–203.
[58] J. McManis, P. Varaiya, Suspension automata: a decidable class of hybrid automata, in: Proceedings of the 6th International Conference on Computer Aided Verification, CAV'94, Springer-Verlag, London, UK, 1994, pp. 105–117.
[59] P.V. Suman, P.K. Pandya, S.N. Krishna, L. Manasa, Timed automata with integer resets: Language inclusion and expressiveness, in: Proceedings of the 6th International Conference on Formal Modeling and Analysis of timed Systems, 2008, pp. 78–92.
[60] L. Manasa, S.N. Krishna, C. Jain, Model checking weighted integer reset timed automata, Theory of Computing Systems 48 (3) (2011) 648–679.
[61] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho, Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems, in: Hybrid Systems, Springer-Verlag, London, UK, 1993, pp. 209–229.
[62] T.A. Henzinger, P.W. Kopke, A. Puri, P. Varaiya, What's decidable about hybrid automata? in: Proceedings of the 27th Annual ACM Symposium on Theory of Computing, STOC'95, ACM, New York, NY, USA, 1995, pp. 373–382.
[63] F. Demichelis, W. Zielonka, Controlled timed automata, in: Proceedings of the 9th International Conference on Concurrency Theory, CONCUR'98, Springer-Verlag, London, UK, 1998, pp. 455–469.
[64] F. Cassez, K.G. Larsen, The impressive power of stopwatches, in: Proceedings of the 11th International Conference on Concurrency Theory, CONCUR'00, Springer-Verlag, London, UK, 2000, pp. 138–152.
[65] C. Dima, R. Lanotte, Distributed time-asynchronous automata, in: Proceedings of the 4th International Conference on Theoretical Aspects of Computing, ICTAC'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 185–200.
[66] S. Akshay, B. Bollig, P. Gastin, M. Mukund, K. Narayan Kumar, Distributed timed automata with independently evolving clocks, in: Proceedings of the 19th International Conference on Concurrency Theory, CONCUR'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 82–97.
[67] B. Bérard, S. Haddad, Interrupt timed automata, in: Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, FOSSACS'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 197–211.
[68] V. Gupta, T.A. Henzinger, R. Jagadeesan, Robust timed automata, in: Proceedings of the International Workshop on Hybrid and Real-Time Systems, Springer-Verlag, London, UK, 1997, pp. 331–345.
[69] R. Alur, S.L. Torre, P. Madhusudan, Perturbed timed automata, in: Hybrid Systems, 2005, pp. 70–85.
[70] G. Behrmann, A. Fehnker, T. Hune, K.G. Larsen, P. Pettersson, J. Romijn, Efficient guiding towards cost-optimality in UPPAAL, in: Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001, Springer-Verlag, London, UK, 2001, pp. 174–188.

[71] K.G. Larsen, J.I. Rasmussen, Optimal reachability for multi-priced timed automata, Theoretical Computer Science 390 (2008) 197–213.
[72] J. Berendsen, D.N. Jansen, J.-P. Katoen, Probably on time and within budget: On reachability in priced probabilistic timed automata, in: Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems, IEEE Computer Society, Washington, DC, USA, 2006, pp. 311–322.
[73] C. Norström, A. Wall, W. Yi, Timed automata as task models for event-driven systems, in: Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications, RTCSA'99, IEEE Computer Society, Washington, DC, USA, 1999, pp. 182–189.
[74] M. Jurdziński, A. Trivedi, Concavely-priced timed automata, in: Proceedings of the 6th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 48–62.
[75] M. Jurdziński, M. Kwiatkowska, G. Norman, A. Trivedi, Concavely-priced probabilistic timed automata, in: Proceedings of the 20th International Conference on Concurrency Theory, CONCUR 2009, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 415–430.
[76] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, L. Tesei, Timed P automata, Electronic Notes Theoretical Computer Science 227 (2009) 21–36.
[77] P. Bouyer, F. Cassez, E. Fleury, K.G. Larsen, Optimal Strategies in Priced Timed Game Automata, in: K. Lodaya, M. Mahajan (Eds.), Proceedings of the 24th Conference on Fundations of Software Technology and Theoretical Computer Science, FSTTCS'04, in: Lecture Notes in Computer Science, vol. 3328, Springer, Chennai, India, 2004, pp. 148–160.
[78] R. Alur, C. Courcoubetis, D. Dill, Model-checking for probabilistic real-time systems (extended abstract), in: Proceedings of the 18th International Colloquium on Automata, Languages and Programming, Springer-Verlag New York, Inc, New York, NY, USA, 1991, pp. 115–126.
[79] M.Z. Kwiatkowska, G. Norman, R. Segala, J. Sproston, Verifying quantitative properties of continuous probabilistic timed automata, in: Proceedings of the 11th International Conference on Concurrency Theory, CONCUR'00, Springer-Verlag, London, UK, 2000, pp. 123–137.
[80] A. Fietzke, H. Hermanns, C. Weidenbach, Superposition-based analysis of first-order probabilistic timed automata, in: Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 302–316.
[81] P. Krcál, W. Yi, Communicating timed automata: The more synchronous, the more difficult to verify, in: Proceedings of the 18th International Conference on Computer Aided Verification, 2006, pp. 249–262.
[82] R. Lanotte, A. Maggiolo-schettini, P. Milazzo, A. Troina, Modeling long-running transactions with communicating hierarchical timed automata, in: Proceedings of the 8th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems, Springer, 2006.
[83] P.S. Pietro, Z. Dang, Automatic verification of multi-queue discrete timed automata, in: Proceedings of the 9th Annual International Conference on Computing and Combinatorics, COCOON'03, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 159–171.
[84] A. Fellah, S. Noureddine, Some succinctness properties of ω-DTAFA, in: Proceedings of the 5th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems, World Scientific and Engineering Academy and Society, Stevens Point, Wisconsin, USA, 2006, pp. 97–103.
[85] F. Wang, Symbolic verification of complex real-time systems with clock-restriction diagram, in: Proceedings of the IFIP TC6/WG6.1 — 21st International Conference on Formal Techniques for Networked and Distributed Systems, FORTE'01, Kluwer, B.V, 2001, pp. 235–250.
[86] O.H. Ibarra, Z. Dang, P.S. Pietro, Verification in loosely synchronous queue-connected discrete timed automata, Theoretical Computer Science 290 (2003) 1713–1735.
[87] J. Hoenicke, Combination of processes, data, and time, Ph.D. Thesis, University of Oldenburg, July 2006.
[88] R. Lanotte, A. Maggiolo-Schettini, A. Peron, Timed cooperating automata, Fundamenta Informaticae 43 (2000) 153–173.
[89] D. Beyer, H. Rust, Cottbus timed automata: formal definition and semantics, in: Proceedings of the 2nd IEEE/IFIP Joint Workshop on Formal Specifications of Computer-Based Systems, 2001, pp. 75–87.
[90] D. D'Souza, M.R. Mohan, Eventual timed automata, in: Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science, 2005, pp. 322–334.
[91] T.A. Henzinger, J.-F. Raskin, P.-Y. Schobbens, The regular real-time languages, in: Proceedings of the 25th International Colloquium on Automata, Languages, and Programming, ICALP'98, Springer-Verlag, London, UK, 1998, pp. 580–591.
[92] D. D'Souza, P.S. Thiagarajan, Product interval automata: a subclass of timed automata, in: Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science, Springer-Verlag, London, UK, 1999, pp. 60–71.
[93] D. D'Souza, N. Tabareau, On timed automata with input-determined guards, in: Proceedings of the Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS 2004) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 2004), 2004, pp. 68–83.
[94] F. Chevalier, D. Deepak D'Souza, P. Prabhakar, On continuous timed automata with input-determined guards, in: Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 369–380.
[95] F. Chevalier, D. D'Souza, P. Prabhakar, Counter-free input-determined timed automata, in: Proceedings of the 5th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 82–97.
[96] N. Tang, M. Ogawa, Event-clock visibly pushdown automata, in: Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 558–569.
[97] Z. Dang, T. Bultan, O.H. Ibarra, R.A. Kemmerer, Past pushdown timed automata and safety verification, Theoretical Computer Science 313 (2004) 57–71.
[98] M. Emmi, R. Majumdar, Decision problems for the verification of real-time software, in: Proceedings of the 9th International Conference on Hybrid Systems: Computation and Control, pp. 200–211, 2006.
[99] A. Trivedi, D. Wojtczak, Recursive timed automata, in: Proceedings of the 8th International Conference on Automated Technology for Verification and Analysis, ATVA'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 306–324.
[100] M. Benerecetti, S. Minopoli, A. Peron, Analysis of timed recursive state machines, in: Proceedings of the 17th International Symposium on Temporal Representation and Reasoning, 2010, pp. 61–68.
[101] S. Bornot, J. Sifakis, S. Tripakis, Modeling urgency in timed systems, in: Revised Lectures from the International Symposium on Compositionality: The Significant Difference, COMPOS'97, Springer-Verlag, London, UK, 1998, pp. 103–129.

[102] S.-W. Lin, P.-A. Hsiung, C.-H. Huang, Y.-R. Chen, Model checking prioritized timed automata, in: Proceedings of the 3rd International Conference on Automated Technology for Verification and Analysis, ATVA'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 370–384.
[103] O.N. Timo, A. Rollet, Conformance testing of variable driven automata, in: Proceedings of the 8th IEEE International Workshop on Factory Communication Systems Communication in Automation, 2010.
[104] R. Barbuti, L. Tesei, Timed automata with urgent transitions, Acta Informatica 40 (2004) 317–347.
[105] S. Lasota, I. Walukiewicz, Alternating timed automata, ACM Transactions on Computational Logic 9 (2008) 10: 1–10: 27.
[106] P. Parys, I. Walukiewicz, Weak alternating timed automata, in: Proceedings of the 36th Internatilonal Colloquium on Automata, Languages and Programming: Part II, ICALP'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 273–284.
[107] O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems (an extended abstract), in: Symposium on Theoretical Aspects of Computer Science, pp. 229–242, 1995.
[108] B. Bérard, P. Gastin, A. Petit, On the power of non-observable actions in timed automata, in: Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, London, UK, 1996, pp. 257–268.
[109] V. Diekert, P. Gastin, A. Petit, Removing epsilon-transitions in timed automata, in: Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science, STACS'97, Springer-Verlag, London, UK, 1997, pp. 583–594.
[110] C. Dima, R. Lanotte, Removing all silent transitions from timed automata, in: Proceedings of the 7th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 118–132.
[111] L. Lamport, A fast mutual exclusion algorithm, ACM Transactions on Computer Systems 5 (1987) 1–11.
[112] IEEE Standard for a High-Performance Serial Bus. IEEE Std 1394–1995, 1996.
[113] L. Bozzelli, S. La Torre, Decision problems for lower/upper bound parametric timed automata, Formal Methods in System Design 35 (2009) 121–151.
[114] R. Alur, K. Etessami, S. La Torre, D. Peled, Parametric temporal logic for "model measuring", ACM Transactions on Computational Logic 2 (2001) 388–407.
[115] V. Bruyère, J.-F. Raskin, Real-time model-checking: Parameters everywhere, Logical Methods in Computer Science 3 (1) (2007).
[116] E.A. Emerson, R.J. Trefler, Parametric quantitative temporal reasoning, in: Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science, LICS'99, IEEE Computer Society, Washington, DC, USA, 1999, pp. 336–343.
[117] F. Wang, Parametric timing analysis for real-time systems, Information and Computation 130 (1996) 131–150.
[118] D. Zhang, R. Cleaveland, Fast on-the-fly parametric real-time model checking, in: Proceedings of the 26th IEEE International Real-Time Systems Symposium, IEEE Computer Society, Washington, DC, USA, 2005, pp. 157–166.
[119] É André, IMITATOR II: A tool for solving the good parameters problem in timed automata, in: Proceedings 12th International Workshop on Verification of Infinite-State Systems, 2010, pp. 91–99.
[120] T.A. Henzinger, P.-H. Ho, H. Wong-Toi, HyTech: a model checker for hybrid systems, in: Proceedings of the 9th International Conference on Computer Aided Verification, CAV '97, Springer-Verlag, London, UK, 1997, pp. 460–463.
[121] F. Wang, REDLIB for the formal verification of embedded systems, in: Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, IEEE Computer Society, Washington, DC, USA, 2006, pp. 341–346.
[122] F. Wang, Efficient verification of timed automata with BDD-like data structures, International Journal on Software Tools for Technology Transfer 6 (2004) 77–97.
[123] M. Knapik, A. Niewiadomski, W. Penczek, A. Pólrola, M. Szreter, A. Zbrzezny, Parametric model checking with VerICS, Transactions on Petri Nets and Other Models of Concurrency 4 (2010) 98–120.
[124] A. Annichini, A. Bouajjani, M. Sighireanu, TReX: a tool for reachability analysis of complex systems, in: Proceedings of the 13th International Conference on Computer Aided Verification, CAV'01, Springer-Verlag, London, UK, 2001, pp. 368–372.
[125] P. Bouyer, F. Chevalier, On conciseness of extensions of timed automata, Journal of Automata, Languages and Combinatorics 10 (2005) 393–405.
[126] P.V. Suman, P.K. Pandya, Determinization and expressiveness of integer reset timed automata with silent transitions, in: Proceedings of the 3rd International Conference on Language and Automata Theory and Applications, LATA'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 728–739.
[127] T.A. Henzinger, P.W. Kopke, State equivalences for rectangular hybrid automata, in: Proceedings of the 7th International Conference on Concurrency Theory, CONCUR'96, Springer-Verlag, London, UK, 1996, pp. 530–545.
[128] R. Ghosh, C. Tomlin, Symbolic reachable set computation of piecewise affine hybrid automata and its application to biological modelling: delta-notch protein signalling, Systems Biology 1 (2004) 170–183.
[129] M. Fränzle, What will be eventually true of polynomial hybrid automata? in: Proceedings of the 4th International Symposium on Theoretical Aspects of Computer Software, TACS'01, Springer-Verlag, London, UK, 2001, pp. 340–359.
[130] L.P. Carloni, R. Passerone, A. Pinto, A.L. Sangiovanni-Vincentelli, Languages and tools for hybrid systems design, Foundations and Trends in Electronic Design Automation 1 (2006) 1–193.
[131] J.M. Davoren, A. Nerode, Logics for hybrid systems, Proceedings of the IEEE 88 (2000) 985–1010.
[132] T.A. Henzinger, The theory of hybrid automata, in: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, IEEE Computer Society, Washington, DC, USA, 1996, p. 278.
[133] G. Labinaz, M.M. Bayoumi, K. Rudie, A survey of modeling and control of hybrid systems, Annual Reviews in Control 21 (1997) 79–92.
[134] C. Tomlin, I. Mitchell, A.M. Bayen, M. Oishi, Computational techniques for the verification of hybrid systems, Proceedings of the IEEE 91 (2003) 986–1001.
[135] S. Tripakis, T. Dang, Model-based Design of Heterogeneous Systems, in: Modeling, Verification and Testing using Timed and Hybrid Automata, CRC Press, 2009 (Chapter).
[136] P. Bouyer, T. Brihaye, V. Bruyère, J.-F. Raskin, On the optimal reachability problem of weighted timed automata, Formal Methods in System Design 31 (2007) 135–175.
[137] G. Behrmann, K.G. Larsen, J.I. Rasmussen, Priced timed automata: algorithms and applications, in: Proceedings of the 3rd International Conference on Formal Methods for Components and Objects, FMCO'04, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 162–182.
[138] K.G. Larsen, Priced timed automata: theory and tools, in: R. Kannan, K.N. Kumar (Eds.), IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, in: Leibniz International Proceedings in Informatics, LIPIcs, vol. 4, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2009, pp. 417–425.

[139] C. Seceleanu, A. Vulgarakis, P. Pettersson, REMES: a resource model for embedded systems, in: Proceedings of the 14th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS'09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 84–94.
[140] D. Ivanov, M. Orlić, C. Seceleanu, A. Vulgarakis, REMES tool-chain: a set of integrated tools for behavioral modeling and analysis of embedded systems, in: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ASE'10, ACM, New York, NY, USA, 2010, pp. 361–362.
[141] T. Brihaye, V. Bruyère, J.-F. Raskin, Model-checking for weighted timed automata, in: Proceedings of the Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS 2004) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 2004), pp. 277–292, 2004.
[142] P. Bouyer, K.G. Larsen, N. Markey, Model-checking one-clock priced timed automata, in: Proceedings of the 10th International Conference on Foundations of Software Science and Computational Structures, FOSSACS'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 108–122.
[143] P. Bouyer, N. Markey, Costs are expensive!, in: J.-F. Raskin, P.S. Thiagarajan (Eds.), Proceedings of the 5th International Conference on Formal Modelling and Analysis of Timed Systems, FORMATS'07, in: Lecture Notes in Computer Science, vol. 4763, Springer, Salzburg, Austria, 2007, pp. 53–68.
[144] J. Berendsen, D.N. Jansen, F.W. Vaandrager, Fortuna: model checking priced probabilistic timed automata, in: Proceedings of the 7th International Conference on the Quantitative Evaluation of Systems, pp. 273–281, 2010.
[145] P. Bouyer, U. Fahrenberg, K.G. Larsen, N. Markey, Quantitative analysis of real-time systems using priced timed automata, Communications of the ACM 54 (2011) 78–87.
[146] P. Krčál, W. Yi, Decidable and undecidable problems in schedulability analysis using timed automata, in: K. Jensen, A. Podelski (Eds.), Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Computer Science, vol. 2988, Springer-Verlag, 2004, pp. 236–250.
[147] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, W. Yi, TIMES — a tool for modelling and implementation of embedded systems, in: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'02, Springer-Verlag, London, UK, 2002, pp. 460–464.
[148] W. Yi, A Calculus of Real Time Systems. Ph.D. thesis, Department of Computer Science, Chalmers University of Technology, 1991.
[149] P. Krčál, M. Stigge, W. Yi, Multi-processor schedulability analysis of preemptive real-time tasks with variable execution times, in: Proceedings of the 5th International Conference on Formal Modeling and Analysis of timed Systems, FORMATS'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 274–289.
[150] G. Păun, Computing with membranes, Journal of Computer and System Sciences 61 (2000) 108–143.
[151] M. Cavaliere, D. Sburlan, Time-independent P systems, in: Proceedings of the 5th International Conference on Membrane Computing, WMC'04, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 239–258.
[152] H.E. Jensen, Model checking probabilistic real time systems, in: Proceedings of the 7th Nordic Workshop on Programming Theory, Chalmers Institute of Technology, 1996, pp. 247–261.
[153] M. Kwiatkowska, G. Norman, R. Segala, J. Sproston, Verifying soft deadlines with probabilistic timed automata, in: Proceedings of the Workshop on Advances in Verification (WAVe 2000), July 2000.
[154] M.Z. Kwiatkowska, G. Norman, J. Sproston, F. Wang, Symbolic model checking for probabilistic timed automata, Information and Computation 205 (7) (2007) 1027–1077.
[155] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: verification of probabilistic real-time systems, in: Proceedings of the 23rd International Conference on Computer Aided Verification, in: LNCS, Springer, 2011.
[156] mcpta. URL http://www.modestchecker.net/.
[157] A.C. Shaw, Communicating real-time state machines, IEEE Transactions on Software Engineering 18 (1992) 805–816.
[158] E. Posse, J. Dingel, Theory and implementation of a real-time extension to the π-calculus, in: Proceedings of the 12th IFIP WG 6.1 International Conference and 30th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Systems, pp. 125–139, 2010.
[159] D. Harel, Statecharts: a visual formalism for complex systems, Science of Computer Programming 8 (3) (1987) 231–274.
[160] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Conallen, K. Houston, Object-Oriented Analysis and Design with Applications, Third Edition, third edition, Addison-Wesley Professional, 2007.
[161] R. Alur, S. Kannan, M. Yannakakis, Communicating hierarchical state machines, in: Proceedings of the 26th International Colloquium on Automata, Languages and Programming, ICAL'99, Springer-Verlag, London, UK, 1999, pp. 169–178.
[162] R. Lanotte, A. Maggiolo-Schettini, P. Milazzo, A. Troina, Design and verification of long-running transactions in a timed framework, Science of Computer Programming 73 (2008) 76–94.
[163] D. Beyer, C. Lewerentz, A. Noack, Rabbit: a tool for BDD-based verification of real-time systems, in: Proceedings of the 15th International Conference on Computer Aided Verification, pp. 122–125, 2003.
[164] R. Lanotte, A. Maggiolo-Schettini, S. Tini, A. Peron, Transformations of timed cooperating automata, Fundamenta Informaticae 47 (2001) 271–282.
[165] D. Drusinsky, D. Harel, On the power of bounded concurrency I: finite automata, Journal of the ACM 41 (1994) 517–539.
[166] C. Dima, Kleene theorems for event-clock automata, in: Proceedings of the 12th International Symposium on Fundamentals of Computation Theory, FCT'99, Springer-Verlag, London, UK, 1999, pp. 215–225.
[167] D. D'Souza, A logical characterisation of event recording automata, in: Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'00, Springer-Verlag, London, UK, 2000, pp. 240–251.
[168] D. D'Souza, A logical characterisation of event clock automata, International Journal Foundations Computer Science 14 (4) (2003) 625–640.
[169] M. Sorea, Tempo: a model-checker for event-recording automata, in: Proceedings of the 1st Workshop on Real-Time Tools, Aalborg, Denmark, August 2001.
[170] W. Penczek, B. Woźna, Towards bounded model checking for Timed Automata, in: L. Czaja (Ed.), Proceedings of the International Workshop on Concurrency, Specification and Programming, CS&P'01, Warsaw University, 2001, pp. 195–209.
[171] R. Alur, P. Madhusudan, Visibly pushdown languages, in: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, ACM Press, 2004, pp. 202–211.
[172] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer, Alternation, Journal of the ACM 28 (1981) 114–133.
[173] M. Dickhöfer, T. Wilke, Timed alternating tree automata: the automata-theoretic solution to the TCTL model checking problem, in: Proceedings of the 26th International Colloquium on Automata, Languages and Programming, ICAL'99, Springer-Verlag, London, UK, 1999, pp. 281–290.

[174] J. Ouaknine, J. Worrell, On the decidability and complexity of metric temporal logic over finite words, Logical Methods in Computer Science 3 (1) (2007).
[175] M. Jenkins, J. Ouaknine, A. Rabinovich, J. Worrell, Alternating timed automata over bounded time, in: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS'10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 60–69.
[176] N.D. Jones, L.H. Landweber, Y.E. Lien, Complexity of some problems in Petri nets, Theoretical Computer Science 4 (1977) 277–299.
[177] B. Gebremichael, F. Vaandrager, Specifying urgency in timed I/O automata, in: Proceedings of the 3rd IEEE International Conference on Software Engineering and Formal Methods, IEEE Computer Society, Washington, DC, USA, 2005, pp. 64–74.
[178] G. Behrmann, A. Cougnard, A. David, E. Fleury, K.G. Larsen, L. Didier, UPPAAL-Tiga: time for playing games!, in: Proceedings of the 19th International Conference on Computer Aided Verification, CAV'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 121–125.
[179] T. Brihaye, T.A. Henzinger, V.S. Prabhu, J. françois Raskin, Minimum-time reachability in timed games, in: Proceedings of the 34th International Colloquium on Automata, Languages and Programming, pp. 825–837, 2007.
[180] T.A. Henzinger, P.W. Kopke, Discrete-time control for rectangular hybrid automata, Theoretical Computer Science 221 (1999) 369–392.
[181] M. Jurdzinski, A. Trivedi, Reachability-time games on timed automata, in: Proceedings of the 34th International Colloquium on Automata, Languages and Programming, pp. 838–849, 2007.
[182] R. Ehlers, R. Mattmüller, H.-J. Peter, Synthia: verification and synthesis for timed automata, in: Proceedings of the 23rd International Conference on Computer Aided Verification, Cliff Lodge, Snowbird, Utah, USA, July 2011.
[183] F. Cassez, N. Markey, Communicating Embedded Systems – Software and Design, Control of Timed Systems, pp. 83–120. 2009 (Chapter).
[184] F. Cassez, T.A. Henzinger, J.-F. Raskin, A comparison of control problems for timed and hybrid systems, in: Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control, HSCC'02, Springer-Verlag, London, UK, 2002, pp. 134–148.
[185] C. Daws, P. Kordy, Symbolic robustness analysis of timed automata, in: E. Asarin, P. Bouyer (Eds.), Proceedings of the 4th International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS'06, in: Lecture Notes in Computer Science, vol. 4202, Springer-Verlag, 2006, pp. 143–155.
[186] M. Swaminathan, M. Franzle, A symbolic decision procedure for robust safety of timed systems, in: Proceedings of the 14th International Symposium on Temporal Representation and Reasoning, IEEE Computer Society, Washington, DC, USA, 2007, p. 192.
[187] C. Dima, Dynamical properties of timed automata revisited, in: Proceedings of the 5th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 130–146.
[188] P. Bouyer, N. Markey, P.-A. Reynier, Robust model-checking of linear-time properties in timed automata, in: J.R. Correa, A. Hevia, M. Kiwi (Eds.), Proceedings of the 7th Latin American Symposium on Theoretical Informatics, LATIN'06, in: Lecture Notes in Computer Science, vol. 3887, Springer, Valdivia, Chile, 2006, pp. 238–249.
[189] R. Jaubert, P.-A. Reynier, Quantitative robustness analysis of flat timed automata, in: Proceedings of the 14th International Conference on Foundations of Software Science and Computational Structures: Part of the joint European Conferences on Theory and Practice of Software, pp. 229–244, 2011.
[190] K.G. Larsen, A. Legay, L.-M. Traonouez, A. Wąsowski, Robust specification of real time components, in: Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 129–144.
[191] J. Ouaknine, J. Worrell, Revisiting digitization, robustness, and decidability for timed automata, in: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society, Washington, DC, USA, 2003, pp. 198–207.
[192] M. Swaminathan, M. Fränzle, J.-P. Katoen, The surprising robustness of (closed) timed automata against clock-drift, in: Proceedings of the 5th IFIP International Conference on Theoretical Computer Science, pp. 537–553, 2008.
[193] K. Altisen, S. Tripakis, Implementation of timed automata: an issue of semantics or modeling?, in: Proceedings of the 3rd International Conference on Formal Modeling and Analysis of Timed Systems, pp. 273–288, 2005.
[194] P.A. Abdulla, P. Krcál, W. Yi, Sampled semantics of timed automata, Logical Methods in Computer Science 6 (3) (2010).
[195] P. Bouyer, K.G. Larsen, N. Markey, O. Sankur, C. Thrane, Timed automata can always be made implementable, in: J.-P. Katoen, B. König (Eds.), Proceedings of the 22nd International Conference on Concurrency Theory, CONCUR'11, in: Lecture Notes in Computer Science, vol. 6901, Springer, Aachen, Germany, 2011.
[196] E. Asarin, O. Maler, A. Pnueli, On discretization of delays in timed automata and digital circuits, in: Proceedings of the 9th International Conference on Concurrency Theory, CONCUR'98, Springer-Verlag, London, UK, 1998, pp. 470–484.
[197] P. Krčál, R. Pelánek, On sampled semantics of timed systems, in: R. Ramanujam, S. Sen (Eds.), Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science, in: Lecture Notes in Computer Science, vol. 3821, Springer-Verlag, 2005, pp. 310–321.
[198] D. Bosscher, I. Polak, F.W. Vaandrager, Verification of an audio control protocol, in: Proceedings of the 3rd International Symposium Organized Jointly with the Working Group Provably Correct Systems on Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer-Verlag, London, UK, 1994, pp. 170–192.
[199] A. Wellings, Concurrent and Real-Time Programming in Java, John Wiley & Sons, 2004.
[200] N. Hakimipour, P.A. Strooper, R. Duke, Exploring model-based development for the verification of real-time Java code, in: Proceedings of the 5th International Verification Workshop in connection with IJCAR 2008, 2008.
[201] N. Hakimipour, P. Strooper, A. Wellings, TART: Timed-automata to real-time java tool, in: Proceedings of the 8th IEEE International Conference on Software Engineering and Formal Methods, SEFM'10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 299–309.
[202] E. Jee, S. Wang, J.K. Kim, J. Lee, O. Sokolsky, I. Lee, A safety-assured development approach for real-time software, in: Proceedings of the 2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA'10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 133–142.
[203] P. Kučera, O. Hynčica, P. Honzík, Implementation of timed automata in a real-time operating system, in: Proceedings of World Congress on Engineering and Computer Science, vol. I, pp. 56–60, October 2010.

[204] M. Ouimet, K. Lundqvist, The TASM toolset: specification, simulation, and formal verification of real-time systems, in: Proceedings of the 19th International Conference on Computer Aided Verification, CAV'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 126–130.
[205] S. Campana, L. Spalazzi, F. Spegni, XAL: a web oriented programming language based on timed-automata, in: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology — Volume 01, IEEE Computer Society, Washington, DC, USA, 2008, pp. 862–868.
[206] M. Lv, N. Guan, Q. Deng, G. Yu, W. Yi, McAiT: a timing analyzer for multicore real-time software, in: Proceedings of the 9th International Conference on Automated Technology for Verification and Analysis, ATVA'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 414–417.
[207] UPPAAL PRO. URL http://www.cs.aau.dk/~arild/uppaal-probabilistic/.
[208] J. Håkansson, J. Carlson, A. Monot, P. Pettersson, D. Slutej, Component-based design and analysis of embedded systems with UPPAAL PORT, in: Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis, ATVA'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 252–257.
[209] A. Hessel, P. Pettersson, CoVer — a real-time test case generation tool, in: Proceedings of the 19th IFIP International Conference on Testing of Communicating Systems and 7th International Workshop on Formal Approaches to Testing of Software, 2007.
[210] K.G. Larsen, M. Mikucionis, B. Nielsen, A. Skou, Testing real-time embedded software using UPPAAL-TRON: an industrial case study, in: Proceedings of the 5th ACM International Conference on Embedded Software, EMSOFT'05, ACM, New York, NY, USA, 2005, pp. 299–306.
[211] P. Krčál, L. Mokrushin, W. Yi, A tool for compositional analysis of timed systems by abstraction (extended abstract), in: E. B. Johnsen, O. Owe, and G. Schneider, (Eds.), Proceedings of the 19th Nordic Workshop on Programming Theory, 2007.
[212] S. Sentilles, A. Pettersson, D. Nystrom, T. Nolte, P. Pettersson, I. Crnkovic, Save-IDE — a tool for design, analysis and implementation of component-based embedded systems, in: Proceedings of the 31st International Conference on Software Engineering, ICSE'09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 607–610.
[213] K. Altisen, S. Tripakis, Tools for controller synthesis of timed systems, in: Proceedings of the 2nd Workshop on Real-Time Tools, July 2002.
[214] E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venter, D. Weil, S. Yovine, TAXYS: a tool for the development and verification of real-time embedded systems, in: Proceedings of the 13th International Conference on Computer Aided Verification, CAV'01, Springer-Verlag, London, UK, 2001, pp. 391–395.
[215] S. Tripakis, C. Courcoubetis, Extending Promela and Spin for real time, in: Proceedings of the 2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems, Springer-Verlag, London, UK, 1996, pp. 329–348.
[216] M. Bozga, S. Graf, I. Ober, I. Ober, J. Sifakis, The IF toolset, in: Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time, in: LNCS, vol. 3185, June 2004.
[217] F. Laroussinie, K.G. Larsen, CMC: a tool for compositional model-checking of real-time systems, in: Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII), FORTE XI / PSTV XVIII'98, Kluwer, B.V, Deventer, The Netherlands, The Netherlands, 1998, pp. 439–456.
[218] S. Kupferschmid, M. Wehrle, B. Nebel, A. Podelski, Faster than UPPAAL? in: Proceedings of the 20th International Conference on Computer Aided Verification, CAV'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 552–555.
[219] J.-Y. Didier, B. Djafri, H. Klaudel, The MIRELA framework: modeling and analyzing mixed reality applications using timed automata, Journal of Virtual Reality and Broadcasting 6 (1) (2009).
[220] M.-E. Cambronero, G. Díaz, V. Valero, E. Martínez, Validation and verification of web services choreographies by using timed automata, Journal of Logic and Algebraic Programming 80 (1) (2011) 25–49.
[221] L. Bendiksen, P.C. Ölveczky, The priced-timed maude tool, in: Proceedings of the 3rd International Conference on Algebra and Coalgebra in Computer Science, CALCO'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 443–448.
[222] G. Madl, N. Dutt, Tutorial for the Open-source DREAM Tool. In CECS Technical Report, 2006.
[223] A. Alfonso, V. Braberman, D. Garbervetsky, N. Kicillof, A. Olivero, F. Schapachnik, VInTiMe: combining high-level finesse with low-level muscle to verify real-time systems, in: Proceedings of the 1st International Conference on Principles of Software Engineering, Buenos Aires, Argentina, October, 2004.
[224] V.A. Braberman, M. Felder, Verification of real-time designs: Combining scheduling theory with automatic formal verification, in: Proceedings of the 7th European Software Engineering Conference held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC/FSE-7, Springer-Verlag, London, UK, 1999, pp. 494–510.
[225] A. Alfonso, V. Braberman, N. Kicillof, A. Olivero, Visual timed event scenarios, in: Proceedings of the 26th International Conference on Software Engineering, ICSE'04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 168–177.
[226] V. Braberman, D. Garbervetsky, A. Olivero, ObsSlice: a timed automata slicer based on observers, in: Proceedings of the 16th International Conference on Computer Aided Verification, in: LNCS, vol. 3114, Springer, 2004, pp. 470–474.
[227] V. Braberman, A. Olivero, F. Schapachnik, Issues in distributed timed model checking: Building Zeus, International Journal on Software Tools for Technology Transfer 7 (2005) 4–18.
[228] K.G. Larsen, J. Pearson, C. Weise, W. Yi, Clock difference diagrams, Nordic Journal of Computing 6 (1999) 271–298.
[229] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, A. Rasse, Data-structures for the verification of timed automata, in: O. Maler (Ed.), Proceedings of the 1997 International Workshop on Hybrid and Real-Time Systems, HART'97, in: Lecture Notes in Computer Science, vol. 1201, Springer-Verlag, 1997, pp. 346–360.
[230] T. Wilke, Automaten und Logiken für zeitabhngige Systeme. Dissertation, Christian-Albrechts-Universität zu Kiel, 1994.