Timed Alternating Tree Automata: The Automata-Theoretic Solution to the TCTL Model Checking Problem

Martin Dickhöfer and Thomas Wilke

[1] Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany
`mdi@informatik.uni-kiel.de`
[2] Lehrstuhl für Informatik VII, RWTH Aachen, 52056 Aachen, Germany
`wilke@informatik.rwth-aachen.de`

Abstract. We introduce timed alternating tree automata as a natural extension of timed automata for the purpose of solving the model checking problem for timed computation tree logic (TCTL) following the automata-theoretic approach. This settles a problem posed by Henzinger, Kupferman, and Vardi.

With their pioneering work in the late fifties and early sixties, Büchi, Rabin, Trakhtenbrot and others demonstrated that automata theory is a powerful tool for studying mathematical theories. Since then automata theory has been successfully applied to a variety of problems in mathematical logic and to numerous logic-related problems in computer science. A direction that has been particularly successful is the application of automata theory to model checking problems; for as different specification logics as LTL, CTL, CTL*, and the μ-calculus, model checking algorithms have been obtained using an automata-theoretic approach, see, for instance, [13,7,12,5]. Although the model checking problem for timed computation tree logic (TCTL) has been known to be decidable (to be precise, PSPACE-complete) for almost a decade now, see, for instance, [1,2], it has withstood a satisfying treatment within an automata-theoretic framework. In fact, as Henzinger, Kupferman, and Vardi point out in the conclusion of [9], an appropriate automata-theoretic framework has not been available. In this paper, we present such a framework for the first time and show how, within this framework, one can derive a model checking algorithm for TCTL. Moreover, the worst-case complexity of the algorithm thus obtained matches the worst-case complexity of previous algorithms.

The automata-theoretic approach to model checking problems can be roughly explained as follows. To check whether or not a given formula φ holds in a system $\boldsymbol{S}$, one first constructs an automaton $\boldsymbol{A}_\varphi$ that accepts the unravelings of all systems in which φ holds, then forms an appropriate product automaton $\boldsymbol{A}_\varphi \times \boldsymbol{S}$, and finally solves a certain word problem (or the emptiness problem) for $\boldsymbol{A}_\varphi \times \boldsymbol{S}$. As pointed out in [9], the particular problem that arises in the timed framework is the following. On the one hand, TCTL satisfiability is undecidable; on the other hand, TCTL model checking is decidable, see [2]. So the automaton model to be used in the timed setting would have to have an undecidable emptiness problem (for automata of the form $\boldsymbol{A}_\varphi$) but some kind of decidable word problem (for automata of the form $\boldsymbol{A}_\varphi \times \boldsymbol{S}$). This is exactly what happens with the automaton model we suggest in this paper: the "one-node acceptance problem" is the particular word problem that we prove to be decidable, but the emptiness problem is undecidable for our automaton model.

The paper is organized as follows. In Section 1, we introduce our new automaton model. In Section 2, we explain what we understand by the one-node acceptance problem and sketch a proof of its decidability. Section 3 is a reminder of TCTL, and Section 4 finally explains how the TCTL model checking problem can be solved in our framework. Due to space limitations, we cannot give a detailed complexity analysis of the constructions presented. To improve readability of the paper, we use a slightly simplified semantics for TCTL and do not address issues related to non-Zenoness. For details about the general setting, the reader is referred to our technical report [6].

For background on timed automata and the automata-theoretic method, we recommend [4] and [5], respectively.

Acknowledgment. The second author is grateful to Hubert Comon, Laboratoire de Spécification et Vérification, ENS Cachan, for many interesting discussions on the topic of this paper.

Notational Conventions. The set of nonnegative integers is denoted by $\mathbf{N}$. The sets of positive real numbers and nonnegative real numbers are denoted by $\mathbf{P}$ and $\mathbf{R}$, respectively.

1 Timed Alternating Tree Automata

This first technical section introduces our model of timed alternating tree automaton. Traditionally, tree automata are devices that are used to define sets of directed trees, see, for instance, [8] or [11,10]. But they can as well be used in a slightly more general way to define sets of rooted directed graphs. (What is important is that their runs are directed trees.) We adopt this more general view, as it turns out to be very convenient when dealing with formulas: a tree automaton can then define the set of all models of a given formula (instead of the set of the unravelings of all models).

Timed graphs. A *timed graph (TG)* over an alphabet A is a triple $\boldsymbol{G} = (W, R, \nu)$ where W is a set of *nodes,* $R \subseteq W \times \mathbf{P} \times W$ is a set of *directed timed edges,* and $\nu\colon W \to A$ is a *labeling.* It is useful to think of an edge (w, d, w') as connecting the nodes w and w' via a direct path of *duration* d. Given a node w, we will write $N(w)$ for $\{(d, w') \mid (w, d, w') \in R\}$, which describes its *neighborhood.*

A *pointed timed graph (PTG)* is a pair $(\boldsymbol{G}, w)$ where $\boldsymbol{G}$ is a timed graph and w a node of $\boldsymbol{G}$. A very simple but useful PTG, which we will later encounter many times, is the *one-node graph,* denoted $\mathbf{G_1}$. It is defined by

$$\mathbf{G_1} = ((\{\emptyset\}, \{\emptyset\} \times \mathbf{P} \times \{\emptyset\}, \nu), \emptyset) \qquad \text{where } \nu(\emptyset) = \emptyset.$$

So the one-node graph is a PTG with one node, denoted $\emptyset$, which is also labeled by $\emptyset$. And for every positive real number there is a self-loop at $\emptyset$ with that particular duration.

A *path* in a TG $\boldsymbol{G}$ or a PTG $(\boldsymbol{G}, w)$ as above is a sequence $w_0, d_0, w_1, d_1, \dots$ such that $(w_i, d_i, w_{i+1}) \in R$ for all $i \geq 0$. Such a path is said to *start* in w_0. A *maximal* path is a path which is either infinite or finite and cannot be extended.

TATA's—informal description. Timed alternating tree automata (TATA's) are equipped with a finite memory and a finite number of clocks to measure distances in time; their behavior is determined by a complex transition function. The "computation" of a TATA on a PTG proceeds in rounds. At the beginning of every round there are several copies of the TATA on different nodes of the PTG; some nodes might be occupied by many copies, others might not accommodate a single one. During a round, each copy splits up in several new copies, which are sent to neighbored nodes and change their states and clock readings on their way according to the transition function. Initially, there is only one copy residing in the distinguished node of the PTG. Acceptance is defined as usual via path conditions.— A formal description follows.

Clock conditions and assignments. The transition functions of TATA's involve so-called clock conditions, which are as with ordinary timed automata and are defined as follows. Let C be a set whose elements are referred to as *clocks* or *clock variables.* The *clock conditions* over C are generated by the grammar

$$\begin{aligned} \langle \mathrm{CC} \rangle &::= \top \mid \bot \mid c \,\#\, n \mid \neg \langle \mathrm{CC} \rangle \mid \langle \mathrm{CC} \rangle \{\vee \mid \wedge\} \langle \mathrm{CC} \rangle \\ \# &::= < \mid \leq \mid > \mid \geq \mid = \mid \neq \end{aligned}$$

where c generates the elements of C and n the elements of $\mathbf{N}$. The set of all clock conditions over C is denoted by $\mathrm{CC}(C)$.

A *clock assignment* over C is a function $C \to \mathbf{R}$, that is, an element of $\mathbf{R}^C$. In a straightforward way, it is defined what it means for $\alpha \in \mathbf{R}^C$ to satisfy $\gamma \in \mathrm{CC}(C)$, denoted $\alpha \models \gamma$. For instance, $\alpha \models c < 5$ if and only if $\alpha(c) < 5$. There are two different kinds of operations on clock assignments that play a role here, one that advances time and another one that resets clocks. Given a clock assignment α and $d \in \mathbf{R}$, $\alpha + d$ denotes the clock assignment α' where $\alpha'(c) = \alpha(c) + d$ for each $c \in C$. Given a clock assignment α and $C' \subseteq C$, $\alpha[C']$ denotes the clock assignment α' where $\alpha'(c) = \alpha(c)$ for $c \notin C'$ and $\alpha'(c) = 0$ for $c \in C'$. By abuse of notation, we use $\mathbf{0}$ to denote the clock assignment that maps every clock to 0 regardless of the underlying set of clocks.

Transition conditions. The values of the transition functions of TATA's are so-called transition conditions, which are defined as follows. Let S be a set whose elements are referred to as *states* and C a set of clocks. The *transition conditions* over S and C are generated by the grammar

$$\langle \mathrm{TC} \rangle ::= \top \mid \bot \mid \langle \mathrm{CC} \rangle \mid \langle \mathrm{TC} \rangle \{\vee \mid \wedge\} \langle \mathrm{TC} \rangle \mid \{\Box \mid \Diamond\}(\langle \mathrm{CC} \rangle, C', s)$$

where s generates the elements of S and C' the subsets of C. The set of all transition conditions over S and C is denoted by $\mathrm{TC}(S, C)$.

Assume we are given a set of states S, a timed graph with set of nodes W, and a set of clocks C. Inductively, we define what it means for $\alpha \in \mathbf{R}^C$, $X \subseteq \mathbf{P} \times W$, and $Y \subseteq W \times \mathbf{R}^C \times S$ to satisfy a transition condition β, denoted $(\alpha, X, Y) \models \beta$. The reader should think of α as a clock assignment assumed by a copy of a TATA in some node of a Kripke structure whose neighborhood is X. The set Y describes what neighbors copies of the TATA are sent to, which state they are in, and what their clock assignment is.

— The boolean constants and connectives are dealt with in the obvious way.

— $(\alpha, X, Y) \models \gamma$ if $\alpha \models \gamma$ (for every clock condition γ).

— $(\alpha, X, Y) \models \Diamond(\gamma, C', s)$ if there exists $(d, w) \in X$ such that $\alpha + d \models \gamma$ and $(w, \alpha + d[C'], s) \in Y$.

— $(\alpha, X, Y) \models \Box(\gamma, C', s)$ if whenever $\alpha + d \models \gamma$ for some $(d, w) \in X$, then $(w, \alpha + d[C'], s) \in Y$.

TATA's—formal definition. We can now give a formal definition of timed alternating tree automata. A *timed alternating tree automaton (TATA)* over an alphabet A is a tuple $\boldsymbol{A} = (S, C, s_I, \tau, F)$ where S is a finite set of *states,* C is a finite set of *clocks,* $s_I \in S$ is an *initial state,* $\tau\colon S \times A \to \mathrm{TC}(S, C)$ is a *transition function,* and F is some *acceptance condition* such as a Büchi condition.

Next, we define runs. Let $\boldsymbol{A}$ be a TATA as above, $\boldsymbol{G} = (W, R, \nu)$ a TG, and assume $\boldsymbol{T} = (V, E, \lambda)$ is a tree with node labels in $W \times \mathbf{R}^C \times S$. Given a node $v \in V$ with $\lambda(v) = (w, \alpha, s)$, we write w_v for w, α_v for α, s_v for s, and L_v for $\{\lambda(v') \mid (v, v') \in E\}$. A node $v \in V$ is *labeled consistently* if $(\alpha_v, N(w_v), L_v) \models \tau(s_v, \nu(w_v))$. Assume u is the root of $\boldsymbol{T}$. Then $\boldsymbol{T}$ is a *run* of $\boldsymbol{A}$ on a PTG $(\boldsymbol{G}, w)$ if $w_u = w$ and every node is labeled consistently. It is said to start with (s_u, α_u).

A *branch* of a run $\boldsymbol{T}$ as above is an infinite path through $\boldsymbol{T}$ starting at the root, i.e., a sequence $v_0, v_1, v_2, \ldots$ where v_0 is the root of $\boldsymbol{T}$ and $(v_i, v_{i+1}) \in E$ for all $i \in \mathbf{N}$. Given such a branch, its *state labeling* is the sequence $s_{v_0}, s_{v_1}, s_{v_2}, \ldots$

A run is accepting if the state labeling of each of its branches is *accepting* with respect to the given acceptance condition F. A run satisfies the *initial condition* if it starts with $(s_I, \mathbf{0})$. It is *successful* when it is accepting and satisfies the initial condition. A PTG $(\boldsymbol{G}, w)$ is *accepted* by a TATA $\boldsymbol{A}$ if there exists a successful run of $\boldsymbol{A}$ on $(\boldsymbol{G}, w)$.

2 The One-Node Acceptance Problem

It is almost folklore that for ordinary finite-state alternating tree automata all standard decision problems such as emptiness, equivalence or containment are decidable. The situation is less satisfactory with timed word automata: emptiness is decidable, but neither equivalence nor containment are, see, for instance, [3]. With timed alternating tree automata, the situation is even less satisfactory: not even emptiness is decidable. But what is decidable is the simplest "word problem" one can imagine, the question whether or not the one-node graph is accepted. This problem is referred to by *one-node acceptance (ONA) problem.*

One-letter alternating ω-automata. To solve the ONA problem (and to determine its complexity), we reduce it to the emptiness problem for "one-letter alternating ω-automata," which has already been solved (and whose complexity has been determined).

We start with a brief review. A *simple transition condition* over a state set S is built up from the elements of S and the boolean constants $\top$ and $\bot$ using $\vee$ and $\wedge$ only. In other words, a simple transition condition of S is a positive boolean combination of elements from $S \cup \{\bot, \top\}$. The set of all these transition conditions is denoted by $\mathrm{STC}(S)$.

A *one-letter alternating ω-automaton (OAOA)* is a tuple $\boldsymbol{A} = (S, s_I, \tau, F)$ where S is a finite set of *states,* s_I is an *initial state,* $\delta \colon S \to \mathrm{STC}(S)$ is a *transition function,* and F is an *acceptance condition.*

Let $\boldsymbol{A}$ be an OAOA as above and $\boldsymbol{T} = (V, E, \lambda)$ a tree with labels in S. A node v of $\boldsymbol{T}$ is labeled consistently when $L_v \models \tau(\lambda(v))$ where $\models$ is defined in the natural way. The tree $\boldsymbol{T}$ is a run if each of its nodes is labeled consistently; it is accepting if each of its branches is accepting according to the acceptance condition F; it is successful if it is accepting and $\lambda(u) = s_I$ for the root u of $\boldsymbol{T}$.

Observe that there is no need to refer to an input here, as these automata are one-letter alternating ω-automata, and over a one-letter alphabet there's only one ω-word, and this particular word is implicit in the above definition of a run.

We are interested in whether or not a successful run exists for a given OAOA. The corresponding decision problem is known as the *one-letter emptiness (OLE) problem for OAOA's.* (Of course, for one-letter ω-automata there's only one word problem, as there is only one word, so the word problem and the emptiness problem are essentially the same.) Regardless of the acceptance condition, the OLE problem is decidable for OAOA's; the complexity of this problem does, however, depend on the acceptance condition under consideration.

Regions. In order to reduce the ONA problem to the OLE problem for OAOA's we factorize clock assignments modulo an appropriate equivalence relation, which goes back to the seminal paper by Alur and Dill, [3], on timed automata. We review the key definitions.

Let C be a finite set of clocks and m a natural number. An equivalence relation between clock assignments over C, denoted $\equiv_m$, is defined as follows. Clock assignments $\alpha, \alpha' \in \mathbf{R}^C$ are said to be *equivalent with respect to m,* denoted $\alpha \equiv_m \alpha'$, if the two conditions below are satisfied.

— For every clock c and every natural number $n \leq m$, $\alpha(c) < n$ iff $\alpha'(c) < n$, as well as $\alpha(c) = n$ iff $\alpha(c) = n$.

— For clocks c, c', if $\alpha(c) \leq m$ and $\alpha(c') \leq m$, then $\mathrm{trc}(\alpha(c)) < \mathrm{trc}(\alpha(c'))$ iff $\mathrm{trc}(\alpha'(c)) < \mathrm{trc}(\alpha'(c'))$. Here, for every $r \in \mathbf{R}$, $\mathrm{trc}(r)$ is the fractional part of r.

For every m, the relation $\equiv_m$ defined in this way is, in fact, an equivalence relation. The set $\mathbf{R}^C/{\equiv_m}$ will be denoted by R_b, and its elements are called *regions.* The equivalence relation $\equiv_m$ has only a finite number of equivalence classes and enjoys quite a number of important properties. Most importantly, it respects advancing clocks (passage of time), resetting clocks, and the satisfaction relation between clock assignments and clock conditions up to the threshold m. For details, see [4].

Below, we will use the following notation. Given a region $r = \alpha/{\equiv_m}$ and a set of clocks C', we will write $r[C']$ for $\alpha[C']/{\equiv_m}$. This is well-defined because of the aforementioned properties of $\equiv_m$. Further, given a region r as above and a clock condition where all clocks are compared only with numbers less than or equal to m, we will write $r \models \gamma$ if and only if $\alpha \models \gamma$. Again, this is well-defined because of the aformentioned properties of $\equiv_m$. The relation $\underset{m}{\rightarrow}$ is defined by

$$\underset{m}{\rightarrow} = \{(\alpha/{\equiv_m}, (\alpha + d)/{\equiv_m}) \mid \alpha \in \mathbf{R}^C,\, d \in \mathbf{P}\} \enspace .$$

The reduction. We can now present the reduction we're interested in. Let $\boldsymbol{A} = (S, C, s_I, \tau, F)$ be a TATA over an alphabet A that includes the empty set. We define an OAOA $\boldsymbol{A}^\rho$—the *discretization* of $\boldsymbol{A}$—such that $\boldsymbol{A}$ accepts the one-node graph $\mathbf{G_1}$ if and only if $\boldsymbol{A}^\rho$ has a successful run. The bounding parameter m we use is the maximum number the clocks from C are compared to in all transition conditions $\tau(s, a)$ with $s \in S$ and $a \in A$. We set

$$\boldsymbol{A}^\rho = (S \times \mathrm{R}_b, (s_I, \mathbf{0}/{\equiv_m}), \tau^\rho, F^\rho)$$

and define τ^ρ and F^ρ as follows.

The transition function τ^ρ maps (s, r) to the formula $\tau(s, \emptyset)^r$ where for every transition condition δ, the formula δ^r is obtained from δ by performing the following substitutions in parallel.

— Every subformula $c\#n$ with $r \models c\#n$ is replaced by $\top$.

— Every subformula $c\#n$ with $r \not\models c\#n$ is replaced by $\bot$.

— Every subformula $\Diamond(\gamma, C', s')$ is replaced by $\displaystyle\bigvee_{r \underset{m}{\rightarrow} r', r' \models \gamma} (s', r'[C'])$.

— Every subformula $\Box(\gamma, C', s')$ is replaced by $\displaystyle\bigwedge_{r \underset{m}{\rightarrow} r', r' \models \gamma} (s', r'[C'])$.

— The acceptance condition F^ρ results from F by an appropriate modification. For instance, if F is a Büchi condition, i. e., F is a subset of S, then $F^\rho = F \times \mathrm{R}_m$.

Theorem 1. *Let $\boldsymbol{A}$ be a TATA. The TATA $\boldsymbol{A}$ accepts the one-node graph $\mathbf{G_1}$ if and only if the OAOA $\boldsymbol{A}^\rho$ has a successful run.*

Since the reduction is computable and the OLE problem for OAOA's is decidable, we immediately get.

Corollary 1. *The ONA problem is decidable.*

To prove the theorem, we turn runs of a TATA $\boldsymbol{A}$ into runs of its discretization $\boldsymbol{A}^\rho$ and vice versa. The constructions needed are more or less straightforward and so are the proofs that they are correct. When non-Zenoness (divergence) is an issue, then the constructions get slightly more complicated while the correctness arguments get involved. For details, see our tech report [6].

3 Timed Computation Tree Logic

In this section, we shortly review syntax and semantics of timed computation tree logic (TCTL). The syntax we use here is from [9]; the semantics is simplified.

Syntax. Let C be a set of clocks and P an arbitrary set whose elements are referred to as *propositions.* The *TCTL formulas* over P and C are generated by the grammar

$$\begin{aligned}\langle\text{TCTL}\rangle ::= \top \mid \bot \mid p \mid \neg p \mid \langle\text{CC}\rangle \mid c.\langle\text{TCTL}\rangle \mid \langle\text{TCTL}\rangle\{\wedge \mid \vee\}\langle\text{TCTL}\rangle \\ \mid \{\mathsf{E} \mid \mathsf{A}\}[\langle\text{TCTL}\rangle\{\mathsf{U} \mid \mathsf{R}\}\langle\text{TCTL}\rangle]\end{aligned}$$

where p generates the elements of P and c the elements of C. The letters U and R are supposed to remind of "until" and "release." Formulas starting with E or A are called *path formulas.* Observe that the above grammar is ambiguous; these ambiguities will, however, not cause any difficulties. Clocks should be viewed as variables, and a prefix like "$c.$" should be viewed as a quantifier binding certain occurrences of the clock variable c.

A TCTL formula φ is said to be *closed* if every occurrence of a clock is bound by some clock quantifier. The set of subformulas of a TCTL formula φ (including φ itself) is denoted by $\text{sub}(\varphi)$. The set of subformulas of φ of the shape $\mathsf{A}[\psi\,\mathsf{U}\,\chi]$ is denoted by $\mathsf{AU}(\varphi)$. Accordingly, $\mathsf{EU}(\varphi)$, $\mathsf{AR}(\varphi)$, and $\mathsf{ER}(\varphi)$ are defined. The set of clocks bound in φ is denoted by $C_b(\varphi)$; formally, a clock c belongs to $C_b(\varphi)$ if there is a formula ψ such that $c.\psi \in \text{sub}(\varphi)$. A TCTL formula φ is in *normal form* if whenever $c.\psi \in \text{sub}(\varphi)$ for some ψ, then $c \notin C_b(\psi)$.

For instance, $c.\mathsf{E}[c.\mathsf{E}[\top\,\mathsf{U}\,c = 5 \wedge p]\,\mathsf{U}\,c = 5 \wedge p]$ is a closed TCTL formula, but it is not in normal form. This formula is, however, equivalent to $c.\mathsf{E}[c'.\mathsf{E}[\top\,\mathsf{U}\,c' = 5 \wedge p]\,\mathsf{U}\,c = 5 \wedge p]$, which is in normal form.

Semantics. TCTL formulas are interpreted in timed Kripke structures, which are timed graphs over alphabets of a special form: a *timed Kripke structure (TKS)* over a set P of *propositional variables* is a timed graph over 2^P, i. e., over the power set of P. A *pointed timed Kripke structure (TPKS)* is a pair $(\boldsymbol{G}, w)$ where $\boldsymbol{G}$ is a timed Kripke structure and w is a node of $\boldsymbol{G}$.

Given a TKS $(\boldsymbol{G}, w)$ over some set P of propositions with $\boldsymbol{G} = (W, R, \nu)$, a clock assignment $\alpha \in \mathbf{R}^C$, and a TCTL formula φ over P and C, we define what it means for φ to hold in $(\boldsymbol{G}, w)$ with respect to α, denoted $(\boldsymbol{G}, w), \alpha \models \varphi$.

— The boolean constants $\top$ and $\bot$ and the boolean connectives $\vee$ and $\wedge$ are dealt with in the obvious way.

— $(\boldsymbol{G}, w), \alpha \models p$ if $p \in \nu(w)$, and $(\boldsymbol{G}, w), \alpha \models \neg p$ if $p \notin \nu(w)$.

— $(\boldsymbol{G}, w), \alpha \models c.\varphi$ if $(\boldsymbol{G}, w), \alpha[\{c\}] \models \varphi$.

— $(\boldsymbol{G}, w), \alpha \models \mathsf{E}[\varphi\,\mathsf{U}\,\psi]$ if there exists a maximal path $w_0, d_0, w_1, d_1, \dots$ starting in w and $n \geq 0$ such that $(\boldsymbol{G}, w_i), \alpha + d_0 + \dots + d_{i-1} \models \varphi$ for $i < n$ and $(\boldsymbol{G}, w_n), \alpha + d_0 + \dots + d_{n-1} \models \psi$.

— $(\boldsymbol{G}, w), \alpha \models \mathsf{A}[\varphi\,\mathsf{U}\,\psi]$ if for every maximal path $w_0, d_0, w_1, d_1, \dots$ starting in w there exists $n \geq 0$ such that $(\boldsymbol{G}, w_i), \alpha + d_0 + \dots + d_{i-1} \models \varphi$ for $i < n$ and $(\boldsymbol{G}, w_n), \alpha + d_0 + \dots + d_{n-1} \models \psi$.

— $(\boldsymbol{G}, w), \alpha \models \mathsf{E}[\varphi\,\mathsf{R}\,\psi]$ if there exists a maximal path $w_0, d_0, w_1, d_1, \dots$ starting in w such that for every $n \geq 0$, $(\boldsymbol{G}, w_n), \alpha + d_0 + \dots + d_{n-1} \models \psi$, unless there exists $i < n$ with $(\boldsymbol{G}, w_i), \alpha + d_0 + \dots + d_{i-1} \models \varphi$.

— $(\boldsymbol{G}, w), \alpha \models \mathsf{A}[\varphi\,\mathsf{R}\,\psi]$ if for every maximal path $w_0, d_0, w_1, d_1, \dots$ starting in w and for every $n \geq 0$, $(\boldsymbol{G}, w_n), \alpha + d_0 + \dots + d_{n-1} \models \psi$, unless there exists $i < n$ with $(\boldsymbol{G}, w_i), \alpha + d_0 + \dots + d_{i-1} \models \varphi$.

It is easy to see that AU and ER are dual to each other just as EU and AR are.

When φ is a closed TCTL formula, then $(\boldsymbol{G}, w), \alpha \models \varphi$ does not depend on α. This is why in these cases we will simply write $(\boldsymbol{G}, w) \models \varphi$ instead of $(\boldsymbol{G}, w), \alpha \models \varphi$ and say that φ holds in $(\boldsymbol{G}, w)$ or that $(\boldsymbol{G}, w)$ is a *model* of φ.

4 The Timed Model Checking Problem

The systems to be model checked are finite-state systems augmented by clocks and clock conditions: a *timed transition system (TTS)* is a tuple (L, C, ν, τ) where L is a finite set of *locations,* C is a finite set of *clocks,* $\nu \colon L \to 2^P$ is a *propositional valuation,* and $\tau \colon L \to 2^{\mathrm{CC}(C) \times 2^C \times L}$ is a *transition function.*

With a TTS $\boldsymbol{S} = (L, C, \nu, \tau)$ we associate the TKS $\boldsymbol{G_S} = (L \times \mathbf{R}^C, R, \nu')$ defined as follows. For all $l \in L$, $\alpha \in \mathbf{R}^C$, $d \in \mathbf{P}$, $C' \subseteq C$, $((l, \alpha), d, (l', \alpha + d[C'])) \in R$ if there exists $\gamma \in \mathrm{CC}(C)$ such that $(\gamma, C', l') \in \tau(l)$ and $\alpha + d \models \gamma$. For all $l \in l$ and $\alpha \in \mathbf{R}^C$, $\nu'(l, \alpha) = \nu(l)$. When $\boldsymbol{S}$ is a TTS, l a location of $\boldsymbol{S}$, and φ a closed TCTL formula, we write $(\boldsymbol{S}, l) \models \varphi$ for $(\boldsymbol{G_S}, (l, \mathbf{0})) \models \varphi$ and say $(\boldsymbol{S}, l)$ is a *model* of φ.

The *timed model checking problem (TMC problem)* is the task to determine for a given TTS $\boldsymbol{S}$ together with a location l and a closed TCTL formula φ whether or not $(\boldsymbol{S}, l)$ is a model of φ.

4.1 Converting TCTL Formulas to TATA's

The first step in solving the timed model checking problem is constructing a TATA $\boldsymbol{A}_\varphi$ for every TCTL formula φ which accepts exactly the models of φ.

Let φ be a TCTL formula over P and C in normal form. We define a TATA $\boldsymbol{A}_\varphi = (\mathrm{sub}(\varphi), C, \varphi, \tau, F)$ over 2^P where the transition function is defined inductively as follows. We set $\tau(p, a) = \tau(\neg p, a) = \top$ for $p \in a$, and, symmetrically, $\tau(\neg p, a) = \tau(p, a) = \bot$, for $p \notin a$. Further, we set $\tau(\psi \vee \chi, a) = \tau(\psi, a) \vee \tau(\chi, a)$ and $\tau(\psi \wedge \chi, a) = \tau(\psi, a) \wedge \tau(\chi, a)$. Finally, we set:

$$
\begin{aligned}
\tau(c.\psi, a) &= \tau(\psi, a) \ , \\
\tau(\gamma, a) &= \gamma \ , \quad \text{for } \gamma \in \mathrm{CC}(C), \\
\tau(\mathsf{E}[\psi \,\mathsf{U}\, \chi], a) &= \tau(\chi, a) \vee (\tau(\psi, a) \wedge \Diamond(\mathsf{E}[\psi \,\mathsf{U}\, \chi], \top, C_b(\mathsf{E}[\psi \,\mathsf{U}\, \chi]))) \ , \\
\tau(\mathsf{A}[\psi \,\mathsf{U}\, \chi], a) &= \tau(\chi, a) \vee (\tau(\psi, a) \wedge \Box(\mathsf{A}[\psi \,\mathsf{U}\, \chi], \top, C_b(\mathsf{A}[\psi \,\mathsf{U}\, \chi])) \\
&\qquad\qquad \wedge \Diamond(\mathsf{A}[\psi \,\mathsf{U}\, \chi], \top, C_b(\mathsf{A}[\psi \,\mathsf{U}\, \chi]))) \ , \\
\tau(\mathsf{E}[\psi \,\mathsf{R}\, \chi], a) &= \tau(\chi, a) \wedge (\tau(\psi, a) \vee \Diamond(\mathsf{E}[\psi \,\mathsf{R}\, \chi], \top, C_b(\mathsf{E}[\psi \,\mathsf{R}\, \chi])) \\
&\qquad\qquad \vee \Box(\mathsf{E}[\psi \,\mathsf{R}\, \chi], \top, C_b(\mathsf{E}[\psi \,\mathsf{R}\, \chi]))) \ , \\
\tau(\mathsf{A}[\psi \,\mathsf{R}\, \chi], a) &= \tau(\chi, a) \wedge (\tau(\psi, a) \vee \Box(\mathsf{A}[\psi \,\mathsf{R}\, \chi], \top, C_b(\mathsf{A}[\psi \,\mathsf{R}\, \chi]))) \ .
\end{aligned}
$$

The acceptance condition of $\boldsymbol{A}_\varphi$ is the Büchi condition given by the set $\mathsf{ER}(\varphi) \cup \mathsf{AR}(\varphi)$.

We prove that the above TATA accepts exactly all models of the given TCTL formula:

Theorem 2. *Let $(\boldsymbol{G}, w)$ be a PTKS and φ a closed TCTL formula in normal form. The PTKS $(\boldsymbol{G}, w)$ is a model of φ if and only if the TATA $\boldsymbol{A}_\varphi$ accepts $(\boldsymbol{G}, w)$.*

The above construction and the proof of the above theorem are similar to the discrete situation as dealt with in [5].

As a consequence of this and the fact that the satisfiability problem for TCTL formulas is undecidable, [2], we note:

Corollary 2. *The emptiness problem for timed alternating tree automata is undecidable.*

4.2 The Product Construction

The second step in solving the timed model checking problem is defining an appropriate product construction that combines a TATA $\boldsymbol{A}$, a TTS $\boldsymbol{S}$, and a location l of $\boldsymbol{S}$ and results in a new TATA which accepts the one-node graph $\mathbf{G_1}$ if and only if $\boldsymbol{A}$ accepts $(\boldsymbol{G_S}, (l, \mathbf{0}))$.

Let $\boldsymbol{A} = (S, C_{\boldsymbol{A}}, s_I, \tau_{\boldsymbol{A}}, F)$ be a TATA, $\boldsymbol{S} = (L, C_{\boldsymbol{S}}, \nu_{\boldsymbol{S}}, \tau_{\boldsymbol{S}})$ a TTS, and l_I a location of $\boldsymbol{S}$. The *product* of $\boldsymbol{A}$ and $(\boldsymbol{S}, l_I)$, simply denoted $\boldsymbol{A} \times (\boldsymbol{S}, l_I)$, is the TATA $(S \times L, C_{\boldsymbol{A}} \cup C_{\boldsymbol{S}}, (s_I, l_I), \tau^*, F^*)$ over $\{\emptyset\}$ where the transition function τ^* maps $((s, l), \emptyset)$ to the formula that is obtained from $\tau_{\boldsymbol{A}}(s, \nu_{\boldsymbol{S}}(l))$ by replacing each occurrence of subformulas of the form $\Diamond(s', \gamma_{\boldsymbol{A}}, C'_{\boldsymbol{A}})$ or $\Box(s', \gamma_{\boldsymbol{A}}, C'_{\boldsymbol{A}})$ by

$$\bigvee_{(\gamma_S, C'_{\boldsymbol{S}}, l') \in \tau(s)} \Diamond((s', l'), \gamma_{\boldsymbol{A}} \wedge \gamma_{\boldsymbol{S}}, C'_{\boldsymbol{A}} \cup C'_{\boldsymbol{S}}) \quad \text{and} \quad \bigwedge_{(\gamma_S, C'_{\boldsymbol{S}}, l') \in \tau(s)} \Box((s', l'), \gamma_{\boldsymbol{A}} \wedge \gamma_{\boldsymbol{S}}, C'_{\boldsymbol{A}} \cup C'_{\boldsymbol{S}}) \ ,$$

respectively. The acceptance condition F^* is obtained from F by an appropriate modification. If, for instance, F is a Büchi condition, then $F^* = F \times L$.

We prove:

Theorem 3. *Let $\boldsymbol{A}$ be a restricted TATA, $\boldsymbol{S}$ a TTS, and l a location of $\boldsymbol{S}$. The TATA $\boldsymbol{A}$ accepts $(\boldsymbol{G_S}, l)$ if and only if the TATA $\boldsymbol{A} \times (\boldsymbol{S}, l)$ accepts the one-node graph $\mathbf{G_1}$.*

As a consequence of this theorem and Theorem 2, we obtain:

Corollary 3. *Let $\boldsymbol{S}$ be a TTS, l a location of $\boldsymbol{S}$, and φ a closed TCTL formula in strict normal form. Then $(\boldsymbol{S}, l)$ is a model of φ if and only if $\boldsymbol{A}_\varphi \times (\boldsymbol{S}, l)$ accepts the one-node graph $\mathbf{G_1}$.*

In view of Corollary 1, we can finally state:

Theorem 4. *The timed model checking problem is decidable.*

A closer analysis shows:

Theorem 5. *The timed model checking problem is in PSPACE.*

A use of more complicated and more sophisticated acceptance conditions (hesitant and Libi conditions, see [9]) in the construction of $\boldsymbol{A}_\varphi$ yields the refined complexity bounds from [9]. The details are given in [6].

5 Conclusion

We have provided an automata-theoretic framework for dealing with timed trees (graphs) that allows a satisfying treatment of the model-checking problem for timed computation tree logic. It is conceivable that within this framework one can study timed bisimulation, synthesis of controllers for real-time systems, and model-checking for timed versions of the modal μ-calculus, just as this is known from the theory of discrete systems.

References

1. R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In LICS '90, Philadelphia, Pennsylvania, pp. 414–425.
2. R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Inform. and Computation*, 104(1):1–34, 1993.
3. R. Alur and D. L. Dill. Automata for modeling real-time systems. In M. S. Paterson, ed., *ICALP '90*, Warwick, England, Springer, LNCS 443, pp. 322–335.
4. R. Alur and D. L. Dill. A theory of timed automata. *Theoret. Comput. Sci.*, 126(2):183–235, 1994.
5. O. Bernholtz [Kupferman], M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In D. L. Dill, ed., *CAV '94*, Stanford, California, Springer, LNCS 818, pp. 142–155.
6. M. Dickhöfer and Th. Wilke. The automata-theoretic method works for TCTL model checking. Techn. Rep. 9811, Inst. f. Informatik u. Prakt. Math., CAU Kiel, 1998. Available as `http://www-i7.informatik.rwth-aachen.de/~wilke/publications/Rep9811.html.ps.gz`.
7. E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In C. Courcoubetis, ed., *CAV '93*, Elounda, Greece, Springer, LNCS 697, pp. 385–396.
8. F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiakó, Budapest, 1984.
9. T. A. Henzinger, O. Kupferman, and M. Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In U. Montanari, V. Sassone, eds., *Concur '96*, Pisa, Italy, Springer, LNCS 1119, pp. 514–529.
10. W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, eds., *Handbook of Formal Languages*, volume 3. Springer, Berlin, 1997, pp. 389–455.
11. W. Thomas. Automata on infinite objects. In J. v. Leeuwen, ed., *Handbook of Theoretical Computer Science*, vol. B, Elsevier, Amsterdam, 1990, pp. 134–191.
12. M. Y. Vardi. Alternating automata and program verification. In J. v. Leeuwen, ed., *Computer Science Today*, Springer, 1995, LNCS 1000, pp. 471–485.
13. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In D. Kozen, ed., *LICS 86,* Cambridge, Mass., pp. 322–331.