A Space-Efficient On-the-fly Algorithm for Real-Time Model Checking

Thomas A. Henzinger[1]* Orna Kupferman[2] Moshe Y. Vardi[3]

[1] UC Berkeley, EECS Department, Berkeley, CA 94720-1770, U.S.A.
Email: tah@eecs.berkeley.edu URL: http://www.eecs.berkeley.edu/~tah

[2] Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, U.S.A.
Email: ok@research.att.com

[3] Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email: vardi@cs.rice.edu URL: http://www.cs.rice.edu/~vardi

Abstract. In temporal-logic model checking, we verify the correctness of a program with respect to a desired behavior by checking whether a structure that models the program satisfies a temporal-logic formula that specifies the behavior. The main practical limitation of model checking is caused by the size of the state space of the program, which grows exponentially with the number of concurrent components. This problem, known as the state-explosion problem, becomes more difficult when we consider *real-time model checking*, where the program and the specification involve quantitative references to time. In particular, when use timed automata to describe real-time programs and we specify timed behaviors in the logic TCTL, a real-time extension of the temporal logic CTL with clock variables, then the state space under consideration grows exponentially not only with the number of concurrent components, but also with the number of clocks and the length of the clock constraints used in the program and the specification. Two powerful methods for coping with the state-explosion problem are *on-the-fly* and *space-efficient* model checking. In on-the-fly model checking, we explore only the portion of the state space of the program whose exploration is essential for determining the satisfaction of the specification. In space-efficient model checking, we store in memory the minimal information required, preferring to spend time on reconstructing information rather than spend space on storing it. In this work we develop an *automata-theoretic* approach to TCTL model checking that combines both methods. We suggest, for the first time, a PSPACE on-the-fly model-checking algorithm for TCTL.

1 Introduction

While *program verification* was always a desirable, but never an easy task, the advent of concurrent programming has made it significantly more necessary and difficult. The first step in program verification is to come up with a *formal specification* of the program. *Temporal logics* can describe a temporal ordering of events and have been adopted as a powerful tool for specifying and verifying concurrent programs [Pnu77, MP92]. In temporal logic *model checking*, we verify that a program meets a desired behavior by checking that a mathematical model of the program satisfies a temporal logic formula that specifies the behavior [CE81, QS81]. Temporal logics come in two varieties: *linear* and *branching* [Lam80]. In linear temporal logics, formulas are interpreted over infinite words and describe the behavior of each of the infinite computations of a program. In branching temporal logics, formulas are interpreted over infinite trees and describe the behavior of the possible computations of a nondeterministic program.

* Supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the AFOSR contract F49620-93-1-0056, and by the ARPA grant NAG2-892.

The development of concurrent programs that must meet rigid *real-time* constraints has brought with it a need for *real-time temporal logics* that enable quantitative reference to time [EMSS90, AH92]. Formulas of real-time temporal logics are interpreted over *timed structures* (infinite words or infinite trees in which a time stamp is associated with every position), and their syntax includes, in addition to qualitative temporal operators (such as "eventually"), also explicit time references (such as "within 4 time units"). Early research on real-time temporal logics considered *discrete time*, where time stamps are *integers*. It turned out that model-checking methods for untimed temporal logics can be extended quite easily to handle discrete time [Eme92, AH93]. Present research on real-time temporal logics focuses on *dense time*, where time stamps are *reals* [Alu91, AFH96]. There, model checking becomes significantly more complex. For example, while the model-checking problem is PSPACE-complete for the linear temporal logic LTL [SC85], it is undecidable for its real-time extension TLTL [AH94].

We can still define real-time temporal logics with a dense-time domain for which model checking is decidable. Such a logic is TCTL, the real-time extension of the branching temporal logic CTL. The branching nature and the limited syntax of TCTL circumvent the difficulties that make TLTL model checking undecidable and yet enables the specifications of many interesting real-time properties. Nevertheless, model checking for TCTL is hard. Indeed, while the model-checking problem for CTL can be solved in linear time [CES86], it is PSPACE-complete for TCTL [ACD93]. This means that methods that cope with the computational limitations of CTL model checking become even more essential when we turn to consider TCTL model checking.

The main computational limitation of CTL model checking is caused by the size of the program. In particular, in a concurrent setting, the state space of the program grows exponentially with the number of its concurrent components. This issue, known as the *state-explosion* problem, is one of the most challenging issues in the area of computer-aided verification and is the subject of active research. Two powerful methods for coping with the state explosion problem are *on-the-fly* model checking and *space-efficient* model checking. In on-the-fly model checking, we explore only a portion of the state space of the program; namely, the portion whose exploration is essential for determining the satisfaction of the specification [FMJJ92]. On-the-fly model-checking is strongly related to *local* model-checking, where we check whether a specific state of the program satisfies a specification [SW89]. The motivation for space-efficient algorithms comes from the fact that space, rather than time, is the computational bottleneck of model checking. Accordingly, in space-efficient model checking, we store in memory the minimal information required, preferring to spend time on reconstructing information rather than spend space on storing it. In this work we develop an automata-theoretic approach to TCTL model checking that combines both methods and suggests, for the first time, a PSPACE on-the-fly model-checking algorithm for TCTL.

We can view temporal-logic formulas as describing languages (of infinite words or trees). *Automata on infinite objects* also describe languages. Like temporal logics, they come in two varieties: automata on infinite words (*word automata*, for short) and automata on infinite trees (*tree automata*). The automata-theoretic approach to temporal logic uses the theory of automata as a unifying paradigm for program specification and verification [VW94, BVW94]. In this paradigm, both the program and the specification are translated to (or are given as) automata. Linear temporal logic formulas correspond to word automata and branching temporal logic formulas correspond to tree automata. Then, questions about programs and their specifications can be reduced to questions about automata. More specifically, questions such as satisfiability of specifications and correctness of programs with respect to their specifications can be reduced to questions such as nonemptiness and containment of automata. These reductions yield clean and optimal algorithms and are helpful in implementing model-checking methods [CVWY92].

In the automata-theoretic approach to CTL model checking, we translate a formula ψ to a tree *hesitant alternating automaton* (HAA) $\mathcal{A}_\psi$ of size linear in the length of ψ. For checking whether a program P satisfies ψ, we check the nonemptiness of the product of $\mathcal{A}_\psi$ and P.

This product is a word HAA over a 1-letter alphabet and its nonemptiness can be tested in linear time, matching the known bound. Moreover, the special structure of HAA enables us to check their nonemptiness in efficient space and leads to an on-the-fly model-checking algorithm for CTL that needs space linear in the length of ψ but only polylogarithmic in the size of P [BVW94]. So, for model checking of untimed temporal logics, the automata-theoretic approach combines on-the-fly methods and space efficiency.

The automata-theoretic counterpart for describing languages of timed objects are *timed automata* [AD94]. Timed automata use finitely many real-valued *clocks* to keep track of timing constraints, and serve as the common way of modeling the behavior of real-time programs. The automata-theoretic approach to CTL cannot be easily extended to handle TCTL. The reason is TCTL's dense time domain which requires models with infinitely many states. It was shown, however, in [AD94], that each real-time program induces a finite quotient of the infinite state space. More precisely, it partitions the infinite time domain of clock valuations into finitely many *regions*, each of which can be viewed as a set of clock constraints (e.g., $2 < clock_1 < 3$; $clock_2 = 1$). This finite quotient enables us to place the problem of TCTL model checking in the framework of untimed automata. In fact, the automata we use are the same alternating automata used for CTL model checking. Given a TCTL formula ψ and a real-time program $\mathcal{U}$, we construct a 1-letter word HAA $\mathcal{D}_{\psi,\mathcal{U}}$ such that the language of $\mathcal{D}_{\psi,\mathcal{U}}$ is not empty iff $\mathcal{U}$ satisfies ψ.

Several model-checking algorithms for TCTL are studied in the literature [ACD93, HNSY94, LL95, SS95]. All these algorithms use the finite quotient of the state space suggested in [AD94]. They differ in how they explore the finite-state structure it induces. In [ACD93], a bottom-up labeling algorithm is used. This basic algorithm gains two optimizations in [HNSY94]. There, the algorithm uses a symbolic presentation of the state space and it tries to reduce the state space by integrating regions together. Typically, regions whose clock values are indistinguishable by the clock constraints in the program and the formula are integrated into a single *zone*. More work on reducing the state space is done in [SS95], adopting the minimization algorithm of [YL93]. The model-checking algorithm TMC, suggested in [SS95], combines top-down and bottom-up reasoning. The algorithm TMC works on-the-fly and explores only a portion of the program checked, but, unlike our algorithm, TMC maintains this portion all along its execution. Thus, the best bound to its complexity is exponential space. Maintenance of this portion and the bottom-up reasoning in [HNSY94, YL93, SS95] are essential, however, for obtaining zones that are as coarse as possible.

The simple combinatorial structure that emerges from our automata-theoretic approach offers several advantages. First, automata separate the logical and the algorithmic components of the model-checking problem. The translation to automata handles the logic, and the nonemptiness test handles the algorithmics. This separation makes both components simple and enables us to identify the exact complexity of the model checking problem. Second, using *Libi alternating automata* (LAA), which are HAA augmented with a fairness condition [KV95], we can restrict path quantification in TCTL to range only over computations of the program that diverge (i.e., in which time always proceed), and we can extend our method to fair-TCTL. Both extensions do not exist in the algorithm TMC. Most importantly, as the nonemptiness test for LAA combines on-the-fly and space efficiency, so does our model-checking algorithm. Thus, we explore only the part of the program whose exploration is essential for determining the satisfaction of ψ, and we do so in space polynomial in the program and in ψ (assuming a binary encoding of the clock constraints).

Optimization is incorporated into the automata-theoretic framework by integrating several transitions of the automaton into a single transition. This achieves only part of the minimization achieved by the algorithm TMC, but enables a pure top-down reasoning, which is the key to our space efficiency. In addition, we point on a subset of TCTL for which our top-down reasoning obtains zones that are as coarse as possible.

2 Definitions

2.1 Timed Structures and Timed Words

We model real-time programs by *timed structures*. Timed structures extend traditional Kripke structures by labeling each transition with a nonnegative real number denoting its duration. Formally, a timed structure is a tuple $K = \langle AP, W, R, w^0, \sigma\rangle$, where

- AP is a finite set of atomic propositions,
- W is a set of states,
- $R \subseteq W \times \mathbb{R} \times W$ is a transition relation (for simplicity, we denote by $\mathbb{R}$ the set of all non negative real numbers),
- $w^0 \in W$ is an initial state,
- $\sigma : W \to 2^{AP}$ maps to each state a set of atomic propositions true in the state.

A *path* in K is an infinite sequence of pairs $\langle w_0, \delta_0\rangle, \langle w_1, \delta_1\rangle, \ldots$, such that for all $i \geq 0$, we have $\langle w_i, \delta_i, w_{i+1}\rangle \in R$. A *timed word* is an infinite sequence $\tau \in (2^{AP} \times \mathbb{R})^\omega$. We sometimes refer to a timed word as a function $\tau : \mathbb{N} \to 2^{AP} \times \mathbb{R}$ and use $\tau_1(i)$ and $\tau_2(i)$ to refer to the i'th event and duration, respectively, in τ. A timed word τ *diverges* iff for all $r \in \mathbb{R}$, there exists an $i \in \mathbb{N}$ such that $r \leq \sum_{j=0}^{i} \tau_2(i)$. Note that each path in a timed structure induces a timed word: the timed word τ^π induced by τ has $\tau^\pi(i) = \langle\sigma(w_i), \delta_i\rangle$ for all $i \in \mathbb{N}$. We say that the path π diverges iff τ^π diverges.

2.2 Linear Timed Automata

We model real-time programs by *linear timed automata*. We now define linear timed automata and the timed structures induced by them.

Given a set C of clocks, a *clock environment* $\mathcal{E} : C \to \mathbb{R}$ assigns to each clock a nonnegative real value. Given a clock environment $\mathcal{E}$, a set $S \subseteq C$ of reset clocks and a time delay $\delta \in \mathbb{R}$, we define $progress(\mathcal{E}, S, \delta)$ as the clock environment $\mathcal{E}'$ where for all $c \in C$, we have

$$\mathcal{E}'(c) = \begin{cases} 0 & \text{if } c \in S, \\ \mathcal{E}(c) + \delta & \text{if } c \notin S. \end{cases}$$

For two clock environments $\mathcal{E}$ and $\mathcal{E}'$, we say that $\mathcal{E} < \mathcal{E}'$ iff for every clock $c \in C$, we have $\mathcal{E}(c) < \mathcal{E}'(c)$. For a set C of clocks, a formula in $guard(C)$ is one of the following:

- **true**, **false**, or $c \sim v$, where $c \in C$, $v \in \mathbb{N}^4$, and $\sim \in \{\geq, >, \leq, <\}$,
- $\theta_1 \vee \theta_2$ or $\theta_1 \wedge \theta_2$, where θ_1 and θ_2 are formulas in $guard(C)$.

A *linear timed automaton* is a tuple $\mathcal{U} = \langle AP, C, L, E, P, inv, l^0\rangle$, where

- AP is a finite set of atomic propositions,
- C is a finite set of clocks,
- L is a finite set of locations,
- $E : L \to 2^{guard(C) \times 2^C \times L}$ is a nondeterministic transition function,
- $P : L \to 2^{AP}$ is a labeling function,
- $inv : L \to guard(C)$ is an invariance function,
- $l^0 \in L$ is an initial location.

[4] As we can multiply all clock values by the same factor [AD94], using $\mathbb{N}$, rather then $\mathbb{Q}$, does not reduce the expressive power of the automata.

A *position* of $\mathcal{U}$ is a pair $\langle l, \mathcal{E} \rangle \in L \times \mathbb{R}^C$. That is, a position describes a location and a clock environment. Given a position $\langle l, \mathcal{E} \rangle$ and a time delay $\delta \in \mathbb{R}$, we say that the position $\langle l', \mathcal{E}' \rangle$ is a δ-*successor* of $\langle l, \mathcal{E} \rangle$ iff there exists a triple $\langle \theta, S, l' \rangle \in E(l)$ such that the following hold:

1. $progress(\mathcal{E}, \emptyset, \delta) \models \theta$,
2. for every $0 \leq \delta' < \delta$, we have $progress(\mathcal{E}, \emptyset, \delta') \models inv(l)$,
3. $\mathcal{E}' = progress(\mathcal{E}, S, \delta)$.

Linear timed automata run on timed words. A *run* r of a linear timed automaton on a timed word is an infinite sequence of positions. Thus, $r \in (L \times \mathbb{R}^C)^\omega$. We sometimes refer to a run as a function $r : \mathbb{N} \to L \times \mathbb{R}^C$. Given a timed word $\tau : \mathbb{N} \to 2^{AP} \times \mathbb{R}$, a run r of $\mathcal{U}$ on τ satisfies the following. For every $i \geq 0$, let $r(i) = \langle l_i, \mathcal{E}_i \rangle$. Then,

- $l_0 = l^0$ and $\mathcal{E}_0$ assigns to all clocks 0,
- for every $i \geq 0$, we have $P(l_i) = \tau_1(i)$,
- for every $i \geq 0$, we have $\langle l_{i+1}, \mathcal{E}_{i+1} \rangle$ is a $\tau_2(i)$-successor of $\langle l_i, \mathcal{E}_i \rangle$.

We say that $\mathcal{U}$ *accepts* a timed word τ iff there exists a run of $\mathcal{U}$ on τ. The *language* of $\mathcal{U}$ is the set of all timed words that $\mathcal{U}$ accepts. Each linear timed automaton induces a timed structure. Formally, the *timed structure of* $\mathcal{U}$ is

$$K_{\mathcal{U}} = \langle AP, L \times \mathbb{R}^C, R, \langle l^0, 0^{|C|} \rangle, \sigma \rangle,$$

where R and σ are defined as follows:

- $\langle \langle l, \mathcal{E} \rangle, \delta, \langle l', \mathcal{E}' \rangle \rangle \in R$ iff $\langle l', \mathcal{E}' \rangle$ is a δ-successor of $\langle l, \mathcal{E} \rangle$,
- for all states $\langle l, \mathcal{E} \rangle \in L \times \mathbb{R}^C$, we have $\sigma(\langle l, \mathcal{E} \rangle) = P(l)$.

Note that the state set of $K_{\mathcal{U}}$ is infinite and that $K_{\mathcal{U}}$ may have an infinite branching degree. It is easy to see that a timed word is accepted by $\mathcal{U}$ iff it is induced by a path in $K_{\mathcal{U}}$.

2.3 The Real-time Branching Temporal Logic TCTL

We specify properties of timed structures using real-time temporal logics. We consider here TCTL, the real-time extension of the branching temporal logic CTL [ACD93]. Formulas of TCTL are defined with respect to the sets AP and $C_{\mathcal{U}}$ of the program's atomic propositions and clocks, respectively, and a set C_ψ of specification clocks. Atomic formulas of TCTL refer to the satisfaction of atomic propositions and put constraints on the values of the clocks. In addition to the Boolean operators $\neg$, $\vee$, and $\wedge$, TCTL allows the time operators U ("until") and $\tilde{U}$ ("dual until"), and the reset quantifier $c.\varphi$, which resets a specification clock c. For example, if p and q are atomic propositions in a linear timed automaton $\mathcal{U}$ with a clock x, then the TCTL formula

$$ApU((x > 5) \wedge (z.A(qU(z = 3))))$$

uses a specification clock z and asserts that in all computations of $\mathcal{U}$, the atom p hold until a point where the value of the clock x is bigger than 5 and the atom q holds along the first 3 time units of all the computations that start at this point. We consider TCTL formulas in a positive normal form in which negation may apply to atomic propositions only. Given $AP, C_{\mathcal{U}}$, and C_ψ, a formula of TCTL is one of the following:

- **true**, **false**, p or $\neg p$, where $p \in AP$,
- $c \sim v$, where $c \in C_{\mathcal{U}} \cup C_\psi$, $v \in \mathbb{N}$, and $\sim \in \{\geq, >, \leq, <\}$,
- $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $E\varphi_1 U \varphi_2$, $A\varphi_1 U \varphi_2$, $E\varphi_1 \tilde{U} \varphi_2$, or $A\varphi_1 \tilde{U} \varphi_2$, where φ_1 and φ_2 are TCTL formulas,

– $c.\varphi$, where $c \in C_\psi$ and φ is a TCTL formula.

The reset quantifier $c.\varphi$ *binds* all free occurrences of c in φ. By renaming clocks we can make sure that no occurrence of a clock in φ is bound to more than one clock. A formula is *closed* if each non-binding occurrence of a specification clock is bound to exactly one clock. We denote by $cl(\psi)$ the set of all subformulas of ψ. It is easy to see that the size of $cl(\psi)$ is bounded by the length of ψ.

TCTL formulas are interpreted over the states of a timed structure $K = \langle AP, W, R, w_0, \sigma\rangle$ and a clock environment $\mathcal{E} : C \to \mathbb{R}$. For $\delta \in \mathbb{R}$, we write $\mathcal{E} + \delta$ to denote the clock environment that maps a clock c to $\mathcal{E}(c) + \delta$. We write $\mathcal{E}[c := 0]$ to denote the clock environment that maps c to 0 and maps every other clock c' to $\mathcal{E}(c')$. Consider a path $\langle w_0, \delta_0\rangle, \langle w_1, \delta_1\rangle \ldots$, in K. Given $\{j, i\} \subseteq \mathbb{N}$ and $\{\delta', \delta\} \subseteq \mathbb{R}$ with $\delta' \leq \delta_j$ and $\delta \leq \delta_i$, we say that $(j, \delta') < (i, \delta)$ iff $j < i$, or $j = i$ and $\delta' < \delta$.

We use $w, \mathcal{E} \models \varphi$ to indicate that a formula φ holds at state w with clock environment $\mathcal{E}$ (with respect to a given timed structure K). The relation $\models$ is defined inductively as follows:

– For all w and $\mathcal{E}$, we have $w, \mathcal{E} \models \mathbf{true}$ and $w, \mathcal{E} \not\models \mathbf{false}$.
– For $p \in AP$, we have $w, \mathcal{E} \models p$ iff $p \in \sigma(w)$ and $w, \mathcal{E} \models \neg p$ iff $p \notin \sigma(w)$.
– For $c \in C$, $v \in \mathbb{N}$, and $\sim \in \{\geq, >, \leq, <\}$, we have $w, \mathcal{E} \models c \sim v$ iff $\mathcal{E}(c) \sim v$.
– $w, \mathcal{E} \models \varphi_1 \vee \varphi_2$ iff $w, \mathcal{E} \models \varphi_1$ or $w, \mathcal{E} \models \varphi_2$.
– $w, \mathcal{E} \models \varphi_1 \wedge \varphi_2$ iff $w, \mathcal{E} \models \varphi_1$ and $w, \mathcal{E} \models \varphi_2$.
– $w, \mathcal{E} \models E\varphi_1 U \varphi_2$ iff there exists a diverging path $\langle w_0, \delta_0\rangle, \langle w_1, \delta_1\rangle \ldots$ in K with $w_0 = w$ such that there exist $i \geq 0$ and $0 \leq \delta \leq \delta_i$ such that $w_i, (\mathcal{E} + \delta + \sum_{j=0}^{i-1} \delta_j) \models \varphi_2$ and for all $(j, \delta') < (i, \delta)$, we have $w_j, (\mathcal{E} + \delta' + \sum_{k=0}^{j-1} \delta_k) \models \varphi_1 \vee \varphi_2$.
– $w, \mathcal{E} \models A\varphi_1 U \varphi_2$ iff for every diverging path $\langle w_0, \delta_0\rangle, \langle w_1, \delta_1\rangle \ldots$ in K with $w_0 = w$, there exist $i \geq 0$ and $0 \leq \delta \leq \delta_i$ such that $w_i, (\mathcal{E} + \delta + \sum_{j=0}^{i-1} \delta_j) \models \varphi_2$ and for all $(j, \delta') < (i, \delta)$, we have $w_j, (\mathcal{E} + \delta' + \sum_{k=0}^{j-1} \delta_k) \models \varphi_1 \vee \varphi_2$.
– $w, \mathcal{E} \models E\varphi_1 \tilde{U} \varphi_2$ iff there exists a diverging path $\langle w_0, \delta_0\rangle, \langle w_1, \delta_1\rangle \ldots$ in K with $w_0 = w$ and for every $i \geq 0$ and $0 \leq \delta \leq \delta_i$ such that $w_i, (\mathcal{E} + \delta + \sum_{j=0}^{i-1} \delta_j) \not\models \varphi_2$, there exists (j, δ') such that $(j, \delta') < (i, \delta)$ and $w_j, (\mathcal{E} + \delta' + \sum_{k=0}^{j-1} \delta_k) \models \varphi_1 \wedge \varphi_2$.
– $w, \mathcal{E} \models A\varphi_1 \tilde{U} \varphi_2$ iff for every diverging path $\langle w_0, \delta_0\rangle, \langle w_1, \delta_1\rangle \ldots$ in K with $w_0 = w$ and for every $i \geq 0$ and $0 \leq \delta \leq \delta_i$ such that $w_i, (\mathcal{E} + \delta + \sum_{j=0}^{i-1} \delta_j) \not\models \varphi_2$, there exists (j, δ') such that $(j, \delta') < (i, \delta)$ and $w_j, (\mathcal{E} + \delta' + \sum_{k=0}^{j-1} \delta_k) \models \varphi_1 \wedge \varphi_2$.
– $w, \mathcal{E} \models c.\varphi$ iff $w, \mathcal{E}[c := 0] \models \varphi$.

Note that path quantification in TCTL ranges over diverging paths only. Note also that the semantics of the until operator and its duality requires either φ_1 or φ_2 to be satisfied until the satisfaction of φ_2. As explained in [HNSY94], this meets our expectations of the dense-time domain. Thus, the formula $z.A(z \leq 4)U(z > 4)$ is always valid and, similarly, the dual formula $z.A(z > 4)\tilde{U}(z \leq 4)$ is not satisfiable.

For a timed structure K and a TCTL formula ψ, we say that $K \models \psi$ iff $w_0, \mathcal{E}_0 \models \psi$, where $\mathcal{E}_0$ denotes the clock environment that assigns 0 to all clocks. The *model-checking* problem for TCTL is defined as follows: given a closed TCTL formula ψ and a linear timed automaton $\mathcal{U}$, determine whether $K_\mathcal{U} \models \psi$.

3 The Model-Checking Method

Given a TCTL formula ψ and a linear timed automaton $\mathcal{U}$, we are going to solve the model-checking problem as follows.

1. Define the set Υ of clock regions induced by ψ and $\mathcal{U}$ [AD94].

2. Define the function $reg\_succ : \Upsilon \to \Upsilon$, that maps a clock region to its (unique) successor clock region [AD94].
3. Construct a 1-letter alternating automaton $\mathcal{D}_{\psi,\mathcal{U}}$ on infinite words. Each state in this untimed automaton is associated with a subformula φ of ψ, a location l of $\mathcal{U}$, and a clock region π from Υ. Intuitively, the language of a state $\langle \varphi, l, \pi \rangle$ is nonempty iff φ is satisfied in all states $\langle l, \mathcal{E} \rangle$ of the timed structure $K_{\mathcal{U}}$ for which the clock environment $\mathcal{E}$ is in π.
4. Check $\mathcal{D}_{\psi,\mathcal{U}}$ for nonemptiness.

We describe our method in two steps. First, in Section 3.4, we ignore the fact that path quantification in TCTL range only over computations that diverge. Then, in Section 3.5, we extend our method to handle path quantification correctly. We now define regions, the function *reg_succ*, and alternating automata.

3.1 Regions and Region Successors

Given a TCTL formula ψ and a linear timed automaton $\mathcal{U}$, let C_ψ and $C_{\mathcal{U}}$ be the set of clocks in ψ and $\mathcal{U}$, respectively. We assume that C_ψ and $C_{\mathcal{U}}$ are disjoint and denote their union by C. The clocks in C run through infinitely many clock environments. We can partition the infinitely many clock environments to finitely many equivalent classes such that all clock environments of the same class are indistinguishable by formulas in $guard(C)$ that are clock constraints in either ψ or $\mathcal{U}$. It was proven in [AD94] that a sufficient condition for two environment clocks to be indistinguishable is agreement on the integral parts of all clocks values and agreement on the ordering of the fractional parts of all clock values. This leads to the following definition of *regions*. For $x \in \mathbb{R}$, let $\lfloor x \rfloor$ and $\langle x \rangle$ denote the integer and the fractional parts of x, respectively. Also, for each $c \in C$, let v_c be the largest integer v for which $x \sim v$ is a subformula of some clock constraint in ψ or $\mathcal{U}$. We define an equivalent relation $\approx \subseteq \mathbb{R}^{|C|} \times \mathbb{R}^{|C|}$. For two clock environments $\mathcal{E}$ and $\mathcal{E}'$, we have that $\mathcal{E} \approx \mathcal{E}'$ iff the following hold:

1. For all $c \in C$, either $\lfloor \mathcal{E}(c) \rfloor = \lfloor \mathcal{E}'(c) \rfloor$, or $\mathcal{E}(c) > v_c$ and $\mathcal{E}'(c) > v_c$.
2. For all $\{c, d\} \subseteq C$ with $\mathcal{E}(c) \leq v_c$ and $\mathcal{E}(d) \leq v_d$, we have that $\langle \mathcal{E}(c) \rangle \leq \langle \mathcal{E}(d) \rangle$ iff $\langle \mathcal{E}'(c) \rangle \leq \langle \mathcal{E}'(d) \rangle$.
3. For all $c \in C$ with $\mathcal{E}(c) \leq v_c$, we have $\langle \mathcal{E}(c) \rangle = 0$ iff $\langle \mathcal{E}'(c) \rangle = 0$.

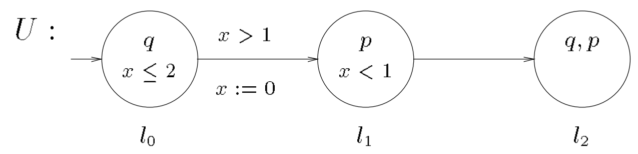
We now define a region as an equivalent class of the relation $\approx$. Let Υ denote the set of all regions induced by ψ and $\mathcal{U}$ and let $rep : \Upsilon \to \mathbb{R}^{|C|}$ map each region to a representative clock environment in it. We represent a region π by $rep(\pi)$. A clock environments $\mathcal{E}$ then belongs to π iff $\mathcal{E} \approx rep(\pi)$. We sometime represent a region also by a finite set of clock constraints (e.g., $[x = 1; 2 < z < 3]$). A clock environment $\mathcal{E}$ then belongs to π iff it satisfies all its clock constraints. Following the definition of regions, the constraints that represent π specify the integral part of all clocks, the order among the fractional parts, and whether they are equal to 0.

Lemma 1. [AD94] *The number of regions in Υ is bounded by $|C|! \cdot 2^{|C|} \cdot \prod_{c \in C}(2v_c + 2)$.*

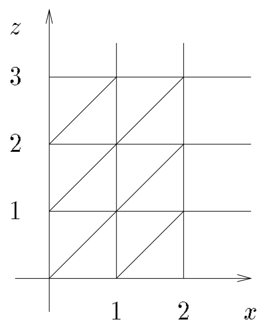
For a region π and a formula $\varphi \in guard(C)$, we say that π satisfies φ (denoted $\pi \models \varphi$) iff $rep(\pi)$ satisfies φ. Note that by the definition of regions, π satisfies φ iff all clock environments in π satisfy φ.

Each region has a unique successor region. Intuitively, the successor of a region π is obtained from π by letting time pass. The function $reg\_succ : \Upsilon \to \Upsilon$ maps a region π to its successor region. For a region π and a clock environment $\mathcal{E}$, we have that $\mathcal{E} \in reg\_succ(\pi)$ iff $rep(\pi) \not\approx \mathcal{E}$, $rep(\pi) < \mathcal{E}$, and for every clock environment $\mathcal{E}'$ with $rep(\pi) < \mathcal{E}' < \mathcal{E}$, we have $\mathcal{E}' \approx rep(\pi)$ or $\mathcal{E}' \approx \mathcal{E}$. So, for example, if π has a clock constraint $c = v$, for $v \neq v_c$, its successor has a clock constraint $v < c < v+1$. If $v = v_c$, then the successor region has a clock constraint $c > v$, reflecting the fact that once c is bigger than v_c, we are no longer interested in how much it is bigger.

Example 1. Consider the TCTL formula $\psi = z.A(q \vee (1 < z < 3))U(p \wedge q)$ and the linear timed automaton $\mathcal{U}$ appearing in the figure below.



The corresponding set Υ of regions, appearing in the figure below, is of size 59.



12 corner points

29 open line segments

18 open regions

Below are two chains of successive regions in Υ.

- $\pi_0 : [x = z = 0] \rightsquigarrow \pi_1 : [0 < x = z < 1] \rightsquigarrow \pi_2 : [x = z = 1] \rightsquigarrow \pi_3 : [1 < x = z < 2] \rightsquigarrow \pi_4 : [x = z = 2] \rightsquigarrow \ldots$
- $\pi_5 : [x = 0; 1 < z < 2] \rightsquigarrow \pi_6 : [0 < x < 1; 1 < z < 2; \langle x \rangle < \langle z \rangle] \rightsquigarrow \pi_7 : [0 < x < 1; z = 2] \rightsquigarrow \pi_8 : [0 < x < 1; 2 < z < 3; \langle z \rangle < \langle x \rangle] \rightsquigarrow \pi_9 : [x = 1; 2 < z < 3] \rightsquigarrow \ldots$

3.2 Alternating Automata

For an introduction to the theory of automata on infinite words see [Tho90]. *Alternating automata* on infinite words generalize nondeterministic automata and were first introduced in [MS87]. Consider a nondeterministic word automaton with a set Q of states and a transition function M. The function M maps an automaton state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a set of states. Each such state suggests a nondeterministic choice for the automaton's next state. For a given set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$, where we also allow the formulas **true** and **false** and, as usual, $\wedge$ has precedence over $\vee$). We can represent the transition function M using $\mathcal{B}^+(Q)$. For example, $M(q, \sigma) = \{q_1, q_2\}$ can be written as $M(q, \sigma) = q_1 \vee q_2$.

In nondeterministic automata, $M(q, \sigma)$ is a disjunction. In alternating automata, $M(q, \sigma)$ can be an arbitrary formula from $\mathcal{B}^+(Q)$. We can have, for instance, a transition $M(q, \sigma) =$

$(q_1 \wedge q_2) \vee q_3$. Thus, several "copies" of the automaton may be sent to the same suffix of the input word. Formally, an *alternating automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is a transition function, $q_0 \in Q$ is an initial state, and α is an acceptance condition (a condition that defines subset of Q^ω). If $|\Sigma| = 1$, then we call $\mathcal{A}$ a 1-letter automaton. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ satisfies θ. A *run* of an alternating automaton $\mathcal{A}$ over an input word $\sigma_0 \cdot \sigma_1 \cdots$ is a tree labeled by elements of Q. The root of the tree is labeled by q_0 and the labels of a node and its children have to satisfy the transition function. That is, if a node x of level i (that is, the path from the root to x is of length i) is labeled q and $\delta(q, \sigma_i) = \theta$, then there is a possibly empty set $Y = \{q_1, q_2, \ldots, q_n\} \subseteq Q$ such that the following hold:

- Y satisfies θ, and
- for all $1 \leq j \leq n$, the node x has a child labeled with q_j.

For example, if $\delta(q_0, \sigma_0) = (q_1 \wedge q_2) \vee q_3$, then possible runs of $\mathcal{A}$ start in a tree with root q_0 and either two children, q_1 and q_2, or a single child q_3. Note that if $\theta = \mathbf{true}$, then x need not have successors. Also, we can not have a run with $\theta = \mathbf{false}$. Note also that each infinite path in the run corresponds to a sequence in Q^ω. The run is accepting if all its infinite paths satisfy the acceptance condition. An automaton is *nonempty* if it accepts some word.

A *hesitant alternating automaton* (HAA) on words [BVW94] is an alternating automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$ where $\alpha = \langle G, B \rangle$ with $G \subseteq Q$ and $B \subseteq Q$ and δ satisfies the following. First, as in *weak alternating automata* [MSS86], there exists a partition of Q into disjoint sets Q_i, and a partial order $\leq$ on the collection of the Q_i's such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma)$ for some $\sigma \in \Sigma$, we have $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. We call this condition the *weakness condition*. In addition, each set Q_i is classified as either *transient*, *existential*, or *universal*, such that for each set Q_i and for all $q \in Q_i$ and $\sigma \in \Sigma$, the following hold:

1. If Q_i is a transient set, then $\delta(q, \sigma)$ contains no elements of Q_i.
2. If Q_i is an existential set, then $\delta(q, \sigma)$ only contains *disjunctively related* elements of Q_i (i.e., if the transition function is rewritten in disjunctive normal form, there is at most one element of Q_i in each disjunct).
3. If Q_i is a universal set, then $\delta(q, \sigma)$ only contains *conjunctively related* elements of Q_i (i.e., if the transition function is rewritten in conjunctive normal form, there is at most one element of Q_i in each conjunct).

We call this condition the *hesitation condition*. It follows that every infinite path of a run of a HAA ultimately gets trapped either within some existential or within some universal set Q_i. For a path ρ, we denote by $\mathit{inf}(\rho)$ the set of states that ρ visits infinitely often. A path ρ satisfies an acceptance condition $\alpha = \langle G, B \rangle$ if and only if either Q_i is an existential set and $\mathit{inf}(\rho) \cap G \neq \emptyset$, or Q_i is a universal set and $\mathit{inf}(\rho) \cap B = \emptyset$. Note that the acceptance condition of HAA combines the Rabin and the Streett acceptance conditions: existential sets refer to the Rabin condition $\{\langle G, \emptyset \rangle\}$ and universal sets refer to the Streett condition $\{\langle B, \emptyset \rangle\}$. The number of sets in the partition of Q is defined as the *depth* of $\mathcal{A}$.

Theorem 2. [BVW94]

(1) *The 1-letter nonemptiness problem for HAA can be solved in linear time.*

(2) *The 1-letter nonemptiness problem for HAA of size n and depth m can be solved in space $O(m \log^2 n)$.*

The restricted structure of HAA suggests an automata-based model-checking algorithm that combines on-the-fly methods with space efficiency [BVW94]. Essentially, HAA enable on-the-fly model-checking since their transition function depends only on the current states and input. They enable space efficiency since their nonemptiness can be checked in efficient space. We now show that the model-checking problem for TCTL can also be placed in the framework of HAA.

3.3 Region Positions and Region-Position Successors

Let $\mathcal{U} = \langle AP, C_{\mathcal{U}}, L, E, P, inv, l^0 \rangle$ be a linear timed automaton. Given a TCTL formula ψ, let Υ and reg_succ be defined as above. For $\pi \in \Upsilon$ and a set S of clocks, we denote by $\pi[S := 0]$ the region obtained from π by reseting all clocks in S. That is, $\pi[S := 0]$ contain the clock environment $progress(rep(\pi), S, 0)$. Also, let π_0 denote the region where all clocks are set to 0. We define a *region position* of the automaton $\mathcal{U}$ as a pair $\langle l, \pi \rangle \in L \times \Upsilon$. When we say that $\mathcal{U}$ is in region position $\langle l, \pi \rangle$, we mean that $\mathcal{U}$ is in location l and that its clock environment is in π. We say that a region position $\langle l, \pi \rangle$ is *admissible* iff π satisfies $inv(l)$.

We know that the automaton $\mathcal{U}$ can be only in admissible region positions. Moreover, when $\mathcal{U}$ is in region position $\langle l, \pi \rangle$, we know what its possible futures are: the automaton $\mathcal{U}$ can either take an *edge transition* and move to another location, possibly resetting some clocks, or take a *time transition* and stay in l while the values of the clocks change and meet the successor region. We now define a function $succ : L \times \Upsilon \to 2^{L \times \Upsilon}$ that, given $\langle l, \pi \rangle$, returns the set of region positions reachable from $\langle l, \pi \rangle$ by either a time or an edge transition. Thus, $\langle l', \pi' \rangle$ is in $succ(\langle l, \pi \rangle)$ iff it is admissible and either $l' = l$ and $\pi' = reg\_succ(\pi)$, or there exists $\langle \theta, S, l' \rangle \in E(l)$ such that π satisfies θ and $\pi' = \pi[S := 0]$.

Example 2. Consider ψ and $\mathcal{U}$ from Example 1. Below is a partial definition of $succ : L \times \Upsilon \to 2^{L \times \Upsilon}$.

- $succ(\langle l_0, \pi_0 \rangle) = \{\langle l_0, \pi_1 \rangle\}$. The only possible next location is l_1. Since π_0 does not satisfy its guard, no edge transition is enabled.
- $succ(\langle l_0, \pi_3 \rangle) = \{\langle l_0, \pi_4 \rangle, \langle l_1, \pi_5 \rangle\}$. Here, both transitions are enabled.
- $succ(\langle l_1, \pi_5 \rangle) = \{\langle l_1, \pi_6 \rangle, \langle l_2, \pi_5 \rangle\}$.
- $succ(\langle l_1, \pi_6 \rangle) = \{\langle l_1, \pi_7 \rangle, \langle l_2, \pi_6 \rangle\}$.
- $succ(\langle l_1, \pi_7 \rangle) = \{\langle l_1, \pi_8 \rangle, \langle l_2, \pi_7 \rangle\}$.
- $succ(\langle l_1, \pi_8 \rangle) = \{\langle l_2, \pi_8 \rangle\}$. Here, as π_9 does not satisfy $inv(l_1)$, a time transition is not enabled.

3.4 The Construction of $\mathcal{A}_{\psi,\mathcal{U}}$

We now describe a construction of $\mathcal{A}_{\psi,\mathcal{U}}$ that ignores the fact that path quantification in TCTL ranges only over computations that diverge. Given ψ and $\mathcal{U}$, we define the HAA $\mathcal{A}_{\psi,\mathcal{U}} = \langle \{a\}, Q, \delta, q_0, \alpha \rangle$, where

- $Q \subseteq cl(\psi) \times L \times \Upsilon$ and $\langle \varphi, l, \pi \rangle \in Q$ iff $\langle l, \pi \rangle$ is admissible (recall that $cl(\psi)$ is the set of all subformulas of ψ).
- $q_0 = \langle \psi, l^0, \pi_0 \rangle$ (if $\langle l^0, \pi_0 \rangle$ is not admissible then the language of $\mathcal{A}_{\psi,\mathcal{U}}$ is clearly empty).
- The transition function δ is given below.
  - $\delta(\langle p, l, \pi \rangle, a) = \begin{cases} \textbf{true} \text{ if } p \in P(l), \\ \textbf{false} \text{ otherwise.} \end{cases}$
  - $\delta(\langle \neg p, l, \pi \rangle, a) = \begin{cases} \textbf{true} \text{ if } p \notin P(l), \\ \textbf{false} \text{ otherwise.} \end{cases}$

- $\delta(\langle c \sim v, l, \pi\rangle, a) = \begin{cases} \textbf{true} \text{ if } \pi \models c \sim v, \\ \textbf{false} \text{ otherwise.} \end{cases}$
- $\delta(\langle \varphi_1 \vee \varphi_2, l, \pi\rangle, a) = \delta(\langle \varphi_1, l, \pi\rangle, a) \vee \delta(\langle \varphi_2, l, \pi\rangle, a)$.
- $\delta(\langle \varphi_1 \wedge \varphi_2, l, \pi\rangle, a) = \delta(\langle \varphi_1, l, \pi\rangle, a) \wedge \delta(\langle \varphi_2, l, \pi\rangle, a)$.
- $\delta(\langle E\varphi_1 U \varphi_2, l, \pi\rangle, a) = \delta(\langle \varphi_2, l, \pi\rangle, a) \vee (\delta(\langle \varphi_1, l, \pi\rangle, a) \wedge \bigvee_{\langle l', \pi'\rangle \in succ(\langle l, \pi\rangle)} \langle E\varphi_1 U \varphi_2, l', \pi'\rangle)$.
- $\delta(\langle A\varphi_1 U \varphi_2, l, \pi\rangle, a) = \delta(\langle \varphi_2, l, \pi\rangle, a) \vee (\delta(\langle \varphi_1, l, \pi\rangle, a) \wedge \bigwedge_{\langle l', \pi'\rangle \in succ(\langle l, \pi\rangle)} \langle A\varphi_1 U \varphi_2, l', \pi'\rangle)$.
- $\delta(\langle E\varphi_1 \tilde{U} \varphi_2, l, \pi\rangle, a) = \delta(\langle \varphi_2, l, \pi\rangle, a) \wedge (\delta(\langle \varphi_1, l, \pi\rangle, a) \vee \bigvee_{\langle l', \pi'\rangle \in succ(\langle l, \pi\rangle)} \langle E\varphi_1 \tilde{U} \varphi_2, l', \pi'\rangle)$.
- $\delta(\langle A\varphi_1 \tilde{U} \varphi_2, l, \pi\rangle, a) = \delta(\langle \varphi_2, l, \pi\rangle, a) \wedge (\delta(\langle \varphi_1, l, \pi\rangle, a) \vee \bigwedge_{\langle l', \pi'\rangle \in succ(\langle l, \pi\rangle)} \langle A\varphi_1 \tilde{U} \varphi_2, l', \pi'\rangle)$.
- $\delta(\langle c.\varphi, l, \pi\rangle, a) = \delta(\langle \varphi, l, \pi[\{c\} := 0]\rangle, a)$.

– The acceptance condition is $\alpha = \langle G, B\rangle$, where G consists of all triples $\langle \varphi, l, \pi\rangle \in Q$ for which φ is of the form $E\varphi_1 \tilde{U} \varphi_2$ and B consists of all triples $\langle \varphi, l, \pi\rangle \in Q$ for which φ is of the form $A\varphi_1 U \varphi_2$. Thus, an accepting run of $\mathcal{A}_{\psi,\mathcal{U}}$ may get trapped in a set associated with a $\tilde{U}$-formula, but may not get trapped in a set associated with an U-formula.

We define a partition of Q into disjoint sets as follows. Each formula $\varphi \in cl(\psi)$ induces a set $Q_\varphi = \{\varphi\} \times L \times \Upsilon$ in the partition. The partial order over the sets is defined by $Q_{\varphi_1} \leq Q_{\varphi_2}$ iff $\varphi_1 \in cl(\varphi_2)$. Since each transition of the automaton from a state associated with φ leads to states associated with formulas in $cl(\varphi)$, the weakness condition holds. We classify the sets as follows.

1. Sets Q_φ for φ of the form $p, \neg p, c \sim v, \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2$, or $c.\varphi$ are transient.
2. Sets Q_φ for φ of the form $E\varphi_1 U \varphi_2$ or $E\varphi_1 \tilde{U} \varphi_2$ are existential.
3. Sets Q_φ for φ of the form $A\varphi_1 U \varphi_2$ or $A\varphi_1 \tilde{U} \varphi_2$ are universal.

It is easy to see that the hesitation condition holds too.

As in the automata for CTL [BVW94], disjunctions and conjunctions in ψ are handled by the existential and universal branching, respectively, of alternating automata. The until operator and its duality are handled by partitioning them to requirements on the present (current location) and requirements on the future (successor locations). The acceptance condition then takes care of requirements that are repeatedly postponed to the future.

Example 3. Consider ψ and $\mathcal{U}$ from Example 1. Below are some transitions of $\mathcal{A}_{\psi,\mathcal{U}}$. Let $\varphi = A(q \vee (1 < z < 3))U(p \wedge q)$.

– $\delta(\langle \psi, l_0, \pi_0\rangle, a) = \delta(\langle \varphi, l_0, \pi_0\rangle, a) = \textbf{false} \vee (\textbf{true} \wedge \langle \varphi, l_0, \pi_3\rangle) = \langle \varphi, l_0, \pi_1\rangle$.
– $\delta(\langle \varphi, l_0, \pi_1\rangle, a) = \langle \varphi, l_0, \pi_2\rangle$.
– $\delta(\langle \varphi, l_0, \pi_2\rangle, a) = \langle \varphi, l_0, \pi_3\rangle$.
– $\delta(\langle \varphi, l_0, \pi_3\rangle, a) = \langle \varphi, l_1, \pi_5\rangle$.
– $\delta(\langle \varphi, l_1, \pi_5\rangle, a) = \langle \varphi, l_1, \pi_6\rangle \wedge \langle \varphi, l_2, \pi_5\rangle$.
– $\delta(\langle \varphi, l_1, \pi_6\rangle, a) = \langle \varphi, l_1, \pi_7\rangle \wedge \langle \varphi, l_2, \pi_6\rangle$.
– $\delta(\langle \varphi, l_1, \pi_7\rangle, a) = \langle \varphi, l_1, \pi_8\rangle \wedge \langle \varphi, l_2, \pi_7\rangle$.
– $\delta(\langle \varphi, l_1, \pi_8\rangle, a) = \langle \varphi, l_2, \pi_5\rangle$.
– $\delta(\langle \varphi, l_2, \pi_5\rangle, a) = \delta(\langle \varphi, l_2, \pi_6\rangle, a) = \delta(\langle \varphi, l_2, \pi_7\rangle, a) = \delta(\langle \varphi, l_2, \pi_8\rangle, a) = \textbf{true}$.

This partial definition suffices to see that there exists a run of $\mathcal{A}_{\psi,\mathcal{U}}$ in which all copies eventually reach **true**. Thus, $\mathcal{A}_{\psi,\mathcal{U}}$ is nonempty.

3.5 Handling Divergence and Fairness

Path quantification in TCTL ranges only over paths that diverge. Consider an infinite sequence $\gamma = \pi_0, \pi_1, \ldots$ of regions. Divergence of γ is defined with respect to the sequence

$\gamma' = rep(\pi_0), rep(\pi_1), \ldots$ of clock environments. We say that γ diverges iff γ' satisfies the following: for every clock c, either c is reset infinitely often or eventually always $c > v_c$, where v_c is the largest constant with which c is compared. This can be specified using the generalized Büchi fairness condition: a sequence of region diverges iff for every clock $c \in C_{\mathcal{U}}$, the sequence visits infinitely often a region with either $c = 0$ or $c > v_c$.

In order to perform *fair* model checking, HAA were extended in [KV95] to *Libi Alternating Automata* (LAA). A LAA is an HAA extended with a generalized Büchi fairness condition $\beta \subseteq 2^Q$; i.e., β is a set of subsets of the state-space of the LAA (the version presented in [KV95] uses a Rabin fairness condition). For a run of a LAA with an acceptance condition α and a fairness condition $\beta = \{F_1, \ldots, F_n\}$, a path ρ that gets trapped within a set Q_i is accepted by the run iff either Q_i is an existential set, in which case ρ satisfies α and for all $1 \leq i \leq n$, we have $inf(\rho) \cap F_i \neq \emptyset$, or Q_i is a universal set, in which case ρ satisfies α or there exists $1 \leq i \leq n$ for which $inf(\rho) \cap F_i = \emptyset$. It is shown in [KV95] that the time and space complexities of the 1-letter nonemptiness problem for HAA apply also to LAA.

We now use LAA in order to range path quantification only over computations that diverge. Given ψ and $\mathcal{U}$, let $\mathcal{A}_{\psi,\mathcal{U}} = \langle \{a\}, Q, \delta, q_0, \alpha \rangle$ be the HAA defined in Section 3.4. We define the LAA $\mathcal{D}_{\psi,\mathcal{U}} = \langle \{a\}, Q', \delta', q_0, \alpha', \beta \rangle$, where

- $Q' = Q \cup (\{E\mathbf{true}, A\mathbf{false}\} \times L \times \Upsilon)$. The new states constitute two new sets. An existential set $Q_E = \{E\mathbf{true}\} \times L \times \Upsilon$, and a universal set $Q_A = \{A\mathbf{false}\} \times L \times \Upsilon$. The task of these sets will get clearer after the definition of α, δ, and β.
- For $\alpha = \langle G, B \rangle$, we have $\alpha' = \langle G \cup Q_E, B \cup Q_A \rangle$.
- The new states are sinks. Thus,
  - $\delta'(\langle E\mathbf{true}, l, \pi \rangle, a) = \bigvee_{\langle l', \pi' \rangle \in succ(\langle l, \pi \rangle)} \langle E\mathbf{true}, l', \pi' \rangle$.
  - $\delta'(\langle A\mathbf{false}, l, \pi \rangle, a) = \bigwedge_{\langle l', \pi' \rangle \in succ(\langle l, \pi \rangle)} \langle A\mathbf{false}, l', \pi' \rangle$.

  In addition, we change δ as follows.
  - $\delta'(\langle E\varphi_1 U \varphi_2, l, \pi \rangle, a) = \delta(\langle E\varphi_1 U \varphi_2, l, \pi \rangle, a) \wedge \bigvee_{\langle l', \pi' \rangle \in succ(\langle l, \pi \rangle)} \langle E\mathbf{true}, l', \pi' \rangle$.
  - $\delta'(\langle A\varphi_1 U \varphi_2, l, \pi \rangle, a) = \delta(\langle A\varphi_1 U \varphi_2, l, \pi \rangle, a) \vee \bigwedge_{\langle l', \pi' \rangle \in succ(\langle l, \pi \rangle)} \langle A\mathbf{false}, l', \pi' \rangle$.
  - $\delta'(\langle E\varphi_1 \tilde{U} \varphi_2, l, \pi \rangle, a) = \delta(\langle E\varphi_1 \tilde{U} \varphi_2, l, \pi \rangle, a) \wedge \bigvee_{\langle l', \pi' \rangle \in succ(\langle l, \pi \rangle)} \langle E\mathbf{true}, l', \pi' \rangle$.
  - $\delta'(\langle A\varphi_1 \tilde{U} \varphi_2, l, \pi \rangle, a) = \delta(\langle A\varphi_1 \tilde{U} \varphi_2, l, \pi \rangle, a) \vee \bigwedge_{\langle l', \pi' \rangle \in succ(\langle l, \pi \rangle)} \langle A\mathbf{false}, l', \pi' \rangle$.

  By the definition of α', if we ignore the fact that $\mathcal{D}_{\psi,\mathcal{U}}$ is a LAA, a copy of the automaton conjunctively sent to a state in Q_E has no significance, it is just like conjuncting a formula with **true**. In a dual way, a copy disjunctively sent to a state in Q_A has no significance either. In LAA, however, these copies are significant. The copy sent to a state in Q_E is guaranteed to accept iff it can continue with a computation that satisfies β. In a dual way, the copy sent to a state in Q_A is guaranteed to reject iff some computation it continues with satisfies β.
- The fairness condition β holds in computations that diverge. For every clock $c \in C_{\mathcal{U}}$, let

$$F_c = \{\langle \varphi, l, \pi \rangle : \pi \models (c = 0) \vee (c > v_c)\}.$$

  Then, $\beta = \{F_c : c \in C_{\mathcal{U}}\}$.

It follows that whenever $\mathcal{D}_{\psi,\mathcal{U}}$ is in a state associated with an existential formula and a location $\langle l, \pi \rangle$, it makes sure that there exists a path starting in $\langle l, \pi \rangle$ that diverges. Similarly, whenever $\mathcal{D}_{\psi,\mathcal{U}}$ is in a state associated with a universal formula and no path that starts in $\langle l, \pi \rangle$ diverges, it does not require satisfaction of the formula. This at first seems too weak, as we want more; we want quantification to range only over diverging paths. To see that it is not too weak, observe that as long as we keep visiting an existential or universal state, we keep sending copies of $\mathcal{D}_{\psi,\mathcal{U}}$ to states in Q_E and Q_A. Consider for example a state associated with the formula $A\varphi_1 U \varphi_2$. Each computation of $\mathcal{D}_{\psi,\mathcal{U}}$ either reaches a region position from which there are no diverging computations (and then, and only then, it can chose the Q_A disjunct), or a region position that satisfies φ_2 (and then, and only then, it can chose the φ_2 disjunct), or continues to visit $A\varphi_1 U \varphi_2$ forever. In the third case, if a path of $K_{\mathcal{U}}$ does not diverge, then it does not satisfy β. Therefore, the fact that it gets trapped in the set $Q_{A\varphi_1 U \varphi_2}$ (and hence does not satisfy α) does not prevent $\mathcal{D}_{\psi,\mathcal{U}}$ from accepting. The explanation is dual for the other cases.

In the definition above, we use β to restrict path quantification to range over paths that diverge. We can add to β more sets (e.g. $\{\langle \varphi, l, \pi \rangle : \mathit{grant} \in P(l)\}$) and restrict path quantification further (in the above example, to paths that diverge and visit a state labeled with *grant* infinitely often). Thus, like fair-CTL of [CES86], we can handle fair-TCTL, where the input linear timed automaton is augmented with fairness constraints.

Theorem 3.

(1) *The size of $\mathcal{D}_{\psi,\mathcal{U}}$ is bounded by $|\psi| \cdot |L| \cdot |C|! \cdot 2^{|C|} \cdot \prod_{c\in C}(2v_c + 2)$.*
(2) *The depth of $\mathcal{D}_{\psi,\mathcal{U}}$ is bounded by $|\psi|$.*
(3) *The automaton $\mathcal{D}_{\psi,\mathcal{U}}$ is nonempty iff $K_{\mathcal{U}} \models \psi$.*

Theorems 3 and 2 imply the following theorem.

Theorem 4. *Given a linear timed automaton $\mathcal{U}$ and a TCTL formula ψ, checking whether $K_{\mathcal{U}} \models \psi$ can be done either*

(1) *in time $O(|\psi| \cdot |L| \cdot |C|! \cdot 2^{|C|} \cdot \prod_{c\in C}(2v_c + 2)))$, or*
(2) *in space $O(|\psi| \cdot \log^2(|\psi| \cdot |L| \cdot |C|! \cdot 2^{|C|} \cdot \prod_{c\in C}(2v_c + 2)))$.*

We note that as the time and space bounds given in Theorem 2 are not obtained simultaneously, so are the above bounds for the TCTL model-checking problem. In particular, the time complexity of our space-efficient algorithm requires more time than the bound in (1).

3.6 Some Optimizations

The bound given in Theorem 4 captures worst-case complexity. In practice, the set of reachable states in $\mathcal{D}_{\psi,\mathcal{U}}$ may be considerably smaller than $|cl(\psi) \times L \times \Upsilon|$. First, not all positions $\langle l, \pi \rangle$ are admissible. Second, as our algorithm works on-the-fly, it does not consider triples that are not relevant for the satisfaction of ψ. In addition, we suggest here an optimization that minimizes the reachable state space further.

Consider a linear timed automaton $\mathcal{U}$. Recall that in each region position, the automaton takes either an edge or a time transition. Practically, it is often the case that only after taking several time transitions one of the following happens: $\mathcal{U}$ is able to take an edge transition (e.g., when all edge transitions are guarded with a lower bound on some clock), or $\mathcal{U}$ reaches a region position whose region satisfies subformulas of ψ that refer to clock values. We would like to integrate these successive time transitions into a single transition.

For $\varphi \in cl(\psi)$ and a region position $\langle l, \pi \rangle$, we define the set $sat(\langle \varphi, l, \pi \rangle)$ of "relevant" clock constraints that π satisfies. A formula θ is in $sat(\langle \varphi, l, \pi \rangle)$ iff $\pi \models \theta$ and either

1. $\theta \in cl(\varphi)$ is of the form $c \sim v$ for $c \in C$, or
2. $\theta \in guard(C)$ and there exist a transition $\langle \theta, S, l' \rangle \in E(l)$ for some S and l'.

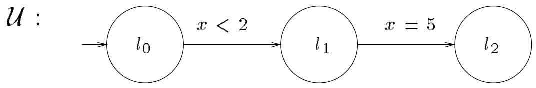
Let $post_l : \Upsilon \to 2^{\Upsilon}$ be the transitive closure of reg_succ when restricted to regions π for which the region position $\langle l, \pi \rangle$ is admissible. That is, the region π' is in $post_l(\pi)$ iff there exists $i > 0$ such that $\pi' = reg\_succ^i(\pi)$ and for all $0 \leq j \leq i$ the pair $\langle l, reg\_succ^j(\pi) \rangle$ is admissible. Given a set $\Upsilon' \subseteq post_l(\pi)$ for some π and l, let $\min \Upsilon'$ denote the region π' for which there exists no

$\pi'' \in \Upsilon'$ with $\pi' \in post_l(\pi'')$. That is, if $\mathcal{U}$ is in region position $\langle l, \pi \rangle$, time passes, and no clocks are reset, then $\min \Upsilon'$ is the first region in Υ' that $\mathcal{U}$ meets.

We can now define a function $opt\_succ : cl(\psi) \times L \times \Upsilon \rightarrow 2^{L \times \Upsilon}$ that, given a formula φ and a region position $\langle l, \pi \rangle$, returns the set of positions reachable from $\langle l, \pi \rangle$ either by an edge transition or by successive time transitions that lead to the first region in which there is a change in satisfaction of relevant clock constraints. Formally, a position $\langle l', \pi' \rangle$ is in $opt\_succ(\langle \varphi, l, \pi \rangle)$ iff it is admissible and either

1. there exists $\langle \theta, S, l' \rangle \in E(l)$ such that $\pi \models \theta$ and $\pi' = \pi[S := 0]$, or
2. $l' = l$ and $\pi' = \min(post_l(\pi) \cap \{\pi'' : sat(\langle \varphi, l, \pi'' \rangle) \neq sat(\langle \varphi, l, \pi \rangle)\})$.

Thus, in opt_succ we integrate together time transitions that go through regions that are equivalent from "the point of view" of the current state of $\mathcal{D}_{\psi,\mathcal{U}}$. Now, whenever a transition $\delta(\langle \varphi, l, \pi \rangle, a)$ contains a disjunction or a conjunction over $succ(\langle l, \pi \rangle)$, we replace it with a disjunction or conjunction over $opt\_succ(\langle \varphi, l, \pi \rangle)$. Some comments about the saving obtained by this integration are in order. Consider the linear timed automaton $\mathcal{U}$ below and assume we model check it with respect to a formula without specification clocks.



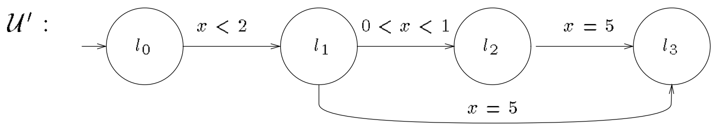
We can split $\mathbb{R}$ into the following four zones.

$$[0 \leq x < 2] \rightsquigarrow [2 \leq x < 5] \rightsquigarrow [x = 5] \rightsquigarrow [x > 5].$$

Each zone forms an equivalence class in the sense that for every location l of $\mathcal{U}$ and two regions π and π' of the same zone, the behavior of $\mathcal{U}$ from the region position $\langle l, \pi \rangle$ is identical to its behavior from the region position $\langle l, \pi' \rangle$. Our algorithm does not achieve such a coarse split of $\mathbb{R}$. Instead, it refines the zone $[0 \leq x < 2]$ into its four regions

$$[x = 0] \rightsquigarrow [0 < x < 1] \rightsquigarrow [x = 1] \rightsquigarrow [1 < x < 2].$$

In this sense, the algorithm in [HNSY94] and the algorithm TMC in [SS95] are more efficient than our algorithm. Why can not we integrate the four regions into a single zone as well? To answer this question, consider the linear timed automaton $\mathcal{U}'$ below.



For $\mathcal{U}'$, the coarsest split of $\mathbb{R}$ into zones equals the split that our algorithm suggests for $\mathcal{U}$. Indeed, we cannot couple together any two regions. When we perform a pure top-down reasoning about $\mathcal{U}$ and $\mathcal{U}'$ (that is, reasoning without look ahead), we discover the difference between them only in location l_1. Therefore, when we integrate regions together, we must get ready to every possible future, in particular to one that requires the split of the zone $[0 \leq x < 2]$. The optimal split performed by [HNSY94, SS95] must involve also some bottom-up reasoning, which increases worst-case space complexity.

From the example, the reader can grasp that the saving achieved by our optimization is quite poor with respect to guards that are upper bounds ($c < v$ or $c \leq v$); there, we can not integrate time transitions. The saving is more significant with respect to guards that are lower bounds ($c > v$ or $c \geq v$); there, we can integrate all the time transitions up to the lower bound. Finally, the saving is optimal (that is, achieves the same coarse split as in [HNSY94, SS95]) with respect to guards that are tight bounds ($c = v$). In particular, it follows that our algorithm is optimal for model checking of the subset of TCTL in which all guards (in both the linear timed automaton and the formula) are tight bounds.

4 Discussion

In this work we suggested a space-efficient on-the-fly algorithm for TCTL model checking. We showed that there is a trade-off between worst-case space efficiency and optimization of the state space by integrating several regions into a zone. The measure for this trade-off is the amount of look-ahead on the input linear timed automaton allowed to the model-checking algorithm. The algorithm in [HNSY94] performs pure bottom-up reasoning (i.e., reasoning with complete look-ahead), and achieves maximal zones. Our algorithm performs pure top-down reasoning (i.e., reasoning without look ahead), and achieves space efficiency. The algorithm TMC in [SS95] combines the two savings; it proceeds on-the-fly to save space, but restricts the look-ahead to only part of the input. Naturally, for each of the algorithms we can contract problems for which the algorithm performs better than the other algorithms. It remains to be seen how the algorithms perform in practice.

Our algorithm is based on an automata-theoretic framework to TCTL model checking. One may wonder why our framework is purposeful for model checking and does not attempt to serve the satisfiability problem. The satisfiability problem for TCTL is undecidable [ACD93]. Hence, a comprehensive automata-theoretic framework for TCTL should involve tree automata for which the nonemptiness problem is undecidable. Such an automata-theoretic framework would contribute significantly to our understanding of real-time temporal logics and would provide a clear explanation to the computational gap between the satisfiability and the model-checking problem of TCTL.

References

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–236, 1994.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[AH92] R. Alur and T.A. Henzinger. Logics and models of real time: a survey. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 74–106. Springer-Verlag, 1992.

[AH93] R. Alur and T. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, May 1993.

[AH94] R. Alur and T.A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[Alu91] R. Alur. *Techniques for Automatic Verification of Real-time Systems.* PhD thesis, Stanford University, 1991.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In D. L. Dill, editor, *Computer Aided Verification, Proc. 6th Int. Conference*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, June 1994. Springer-Verlag.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[Eme92] E.A. Emerson. Real time and the μ-calculus. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 176–194. Springer-Verlag, 1992.

[EMSS90] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Proc. 2nd Workshop on Computer Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 136–145. Springer-Verlag, 1990.

[FMJJ92] J.-C. Fernandez, L. Mounier, C. Jard, and T. Jeron. On-the-fly verification of finite transition systems. *Formal Methods in System Design*, 1:251–273, 1992.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[KV95] O. Kupferman and M.Y. Vardi. On the complexity of branching modular model checking. In *Proc. 6th Conferance on Concurrency Theory*, pages 408–422, Philadelphia, August 1995.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, pages 174–185, January 1980.

[LL95] F. Laroussinie and K. G. Larsen. Compositional model checking of real time systems. In *Proc. 6th Conferance on Concurrency Theory*, pages 27–41, Philadelphia, August 1995.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, Berlin, January 1992.

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54,:267–276, 1987.

[MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Int. Colloquium on Automata, Languages and Programming*. Springer-Verlag, 1986.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symposium on Foundation of Computer Science*, pages 46–57, 1977.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, volume 137, pages 337–351. Springer-Verlag, Lecture Notes in Computer Science, 1981.

[SC85] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *J. ACM*, 32:733–749, 1985.

[SS95] O.V. Sokolsky and S.A. Smolka. Local model checking for real-time systems. In *Computer Aided Verification, Proc. 7th Int. Workshop*, Lecture Notes in Computer Science 939, pages 211–224, Liege, July 1995.

[SW89] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In *Proc. 15th Col. on Trees in Algebra and Programming*. Lecture Notes in Computer Science, 1989.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[YL93] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In C. Courcoubetis, editor, *Computer Aided Verification, Proc. 5th Int. Workshop*, volume 697 of *Lecture Notes in Computer Science 697*, pages 210–224, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.