Distributed Timed Automata with Independently Evolving Clocks[⋆]

S. Akshay[1,3], Benedikt Bollig[1], Paul Gastin[1],
Madhavan Mukund[2], and K. Narayan Kumar[2]

[1] LSV, ENS Cachan, CNRS, France
{akshay,bollig,gastin}@lsv.ens-cachan.fr
[2] Chennai Mathematical Institute, Chennai, India
{madhavan,kumar}@cmi.ac.in
[3] The Institute of Mathematical Sciences, Chennai, India

Abstract. We propose a model of distributed timed systems where each component is a timed automaton with a set of local clocks that evolve at a rate independent of the clocks of the other components. A clock can be read by any component in the system, but it can only be reset by the automaton it belongs to.

There are two natural semantics for such systems. The *universal* semantics captures behaviors that hold under any choice of clock rates for the individual components. This is a natural choice when checking that a system always satisfies a positive specification. However, to check if a system avoids a negative specification, it is better to use the *existential* semantics—the set of behaviors that the system can possibly exhibit under some choice of clock rates.

We show that the existential semantics always describes a regular set of behaviors. However, in the case of universal semantics, checking emptiness turns out to be undecidable. As an alternative to the universal semantics, we propose a *reactive* semantics that allows us to check positive specifications and yet describes a regular set of behaviors.

1 Introduction

In today's world, it is becoming increasingly important to look at networks of timed systems, which allow real-time systems to operate in a distributed manner. Many real-life systems, such as mobile phones, computer servers, and railway crossings, depend crucially on timing while usually consisting of many interacting systems. In general, there is no reason to assume that different timed systems in the networks refer to the same time or evolve at the same rate.

Timed automata [2] are a well-studied formalism to describe systems that require timing. However, networks of timed automata, under the assumption of knowledge of a global time, as done in [5, 10], do not really reflect the distributed model. In this paper, we provide a framework to look at distributed systems with independently evolving local clocks. Each constituent system is modeled by a timed automaton. All clocks belonging to this timed automaton evolve at the same rate. However clocks belonging

⋆ Partially supported by ARCUS, DOTS (ANR-06-SETIN-003), and P2R MODISTE-COVER/RNP Timed-DISCOVERI.

to different processes are allowed to evolve at rates that are independent of each other. We allow clocks belonging to one process to be read/checked by another but we require that a clock can only be reset by the automaton it belongs to. In addition, since we have differing time values on different processes, we are interested in the underlying untimed behaviors of these distributed timed automata rather than their timed behaviors. Thus, the clocks (and time itself) are implementation or synchronization tools rather than being a part of the observation. To ensure that we focus on this problem of varying local time rates, we move to a more general setting with shared memory, which allows us to describe more general systems such as networks of timed asynchronous automata.

It is now natural to look at different semantics depending on the specifications that we want our system to satisfy. When we want to guarantee that our system exhibits a positive specification, we look at the *universal* semantics. This semantics describes the behaviors exhibited by the system no matter how time evolves in the constituent processes. However, if we want to check that our system avoids a negative specification, then we prefer to look at the *existential* semantics. This is the set of behaviors that the system might exhibit under some (bad) choice of local time rates in the constituent processes. We perform a region construction on our distributed timed automata to show that the existential semantics always gives a regular set of untimed behaviors. Thus the model checking problem of distributed timed automata against regular negative specifications is decidable as well. On the other hand, we show that checking emptiness for the universal semantics is undecidable. This is done by a reduction from Post's correspondence problem (PCP) by encoding a PCP instance in terms of the local time rates and ensuring that there is a solution to the PCP instance if and only if there is a valid behavior under all local time rates. This result is further strengthened to a bounded case, where we have restrictions on the relative time rates. Finally, to be able to synthesize and check for positive specifications, we introduce a more intuitive *reactive* semantics, which has the additional advantage of ensuring decidability. This model corresponds to being able to make sure, step by step, that a positive specification is exhibited by our system. This is formally done by defining an equivalent alternating automaton, generating a regular behavior.

Related work. In [6, 12], classical timed automata are equipped with an additional parameter $\varDelta$, which allows a clock to diverge over a period t from its actual value by $\varDelta t$. This model conforms, in a sense, to our existential semantics, where we restrict the set of clock rates to those corresponding to $\varDelta$ (see Section 5). Syntactically, our model coincides with that from [7]: A clock can only be reset by the owner process, whereas it can be read by any process. However, the above works differ from ours since they consider timed words rather than untimed languages. This also explains why our automata differ from hybrid automata [9]. In the model of [3], clocks are not shared and clocks on different processes drift only as long as the processes do not communicate. These assumptions make partial-order–reduction techniques applicable. A fundamental difference between all these approaches and our work is that we do not restrict to system configurations that can be reached under *some* local-time behavior. We also tackle the problem of checking positive specifications by providing semantics that can check if a system exhibits some behavior under *all* relative clock speeds.

Structure of the paper. In Section 2, we introduce our distributed automaton model with independently evolving clocks, and define its existential and universal semantics. Section 3 extends the regions of a timed automaton to our distributed setting, allowing us to compute a finite automaton recognizing the existential semantics. Section 4 shows that checking emptiness of the universal semantics is undecidable. This result is sharpened towards bounded clock drifts in Section 5. Section 6 deals with the reactive semantics, and Section 7 identifies some directions for future work.

A full version of this paper is available [1].

2 Distributed Timed Automata

Preliminaries. For a set Σ, we let Σ^* and Σ^ω denote the set of finite and, respectively, infinite words over Σ. The empty word is denoted by ε. We set $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. The concatenation of words $u \in \Sigma^*$ and $v \in \Sigma^\infty$ is denoted by $u \cdot v$. An *alphabet* is a non-empty finite set. Given an alphabet Σ, we denote by Σ_ε the set $\Sigma \uplus \{\varepsilon\}$. The set of non-negative real numbers is denoted by $\mathbb{R}_{\geq 0}$. For $t \in \mathbb{R}_{\geq 0}$, $\lfloor t \rfloor$ and $\mathit{fract}(t)$ refer to the integral and, respectively, fractional part of t, i.e., $t = \lfloor t \rfloor + \mathit{fract}(t)$.

The set $\mathrm{Form}(\mathcal{Z})$ of *clock formulas* over a set of clocks $\mathcal{Z}$ is given by the grammar $\varphi ::= \mathrm{true} \mid \mathrm{false} \mid x \bowtie C \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$ where x is a clock from $\mathcal{Z}$, $\bowtie \in \{<, \leq, >, \geq, =\}$, and C ranges over $\mathbb{N}$. A *clock valuation* over $\mathcal{Z}$ is a mapping $\nu : \mathcal{Z} \to \mathbb{R}_{\geq 0}$. We say that ν satisfies $\varphi \in \mathrm{Form}(\mathcal{Z})$, written $\nu \models \varphi$, if φ evaluates to true using the values given by ν. For $R \subseteq \mathcal{Z}$, $\nu[R]$ denotes the clock valuation defined by $\nu[R](x) = 0$ if $x \in R$ and $\nu[R](x) = \nu(x)$, otherwise.

The model. Let us recall the fundamental notion of timed automata [2]. These will constitute the building blocks of our distributed timed automata. A *timed automaton* is a tuple $\mathcal{A} = (S, \Sigma, \mathcal{Z}, \delta, I, \iota, F)$ where S is a finite set of *states*, Σ is the alphabet of *actions*, $\mathcal{Z}$ is a finite set of *clocks*, $\delta \subseteq S \times \Sigma_\varepsilon \times \mathrm{Form}(\mathcal{Z}) \times 2^{\mathcal{Z}} \times S$ is the finite set of *transitions*, $I : S \to \mathrm{Form}(\mathcal{Z})$ associates with each state an *invariant*, $\iota \in S$ is the *initial state*, and $F \subseteq S$ is the set of *final states*. We let $\mathit{Reset}(\mathcal{A}) = \{x \in \mathcal{Z} \mid$ there is $(s, a, \varphi, R, s') \in \delta$ such that $x \in R\}$ be the set of clocks that might be reset in $\mathcal{A}$. Without loss of generality, we will assume in this paper that $I(\iota)$ is satisfied by the clock valuation over $\mathcal{Z}$ that maps each clock to 0.

We will now extend the above definition to a distributed setting. First, we fix a non-empty finite set *Proc* of processes (unless otherwise stated). For a tuple t that is indexed by *Proc*, t_p refers to the projection of t onto $p \in \mathit{Proc}$.

Definition 1. *A* distributed timed automaton (DTA) *over the set of processes Proc is a structure $\mathcal{D} = ((\mathcal{A}_p)_{p \in \mathit{Proc}}, \pi)$ where the $\mathcal{A}_p = (S_p, \Sigma_p, \mathcal{Z}_p, \delta_p, I_p, \iota_p, F_p)$ are timed automata and π is a mapping from $\bigcup_{p \in \mathit{Proc}} \mathcal{Z}_p$ to Proc such that, for each $p \in \mathit{Proc}$, we have $\mathit{Reset}(\mathcal{A}_p) \subseteq \pi^{-1}(p) \subseteq \mathcal{Z}_p$.*

Note that $\mathcal{Z}_p$ is the set of clocks that might occur in the timed automaton $\mathcal{A}_p$, either as clock guard or reset. The same clock may occur in both $\mathcal{Z}_p$ and $\mathcal{Z}_q$, since it may be read as a guard in both processes. However, any clock evolves according to the time evolution of some particular process. This clock is then said to *belong* to that process,

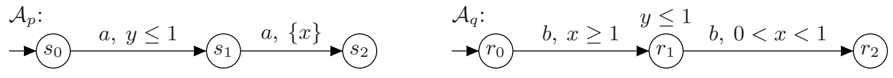


Fig. 1. A distributed timed automaton over $\{p, q\}$

and the *owner* map, π, formalizes this in the above definition. This will become more clear when we describe the formal semantics later in this section. Further, we assume that a clock can only be reset by the process it belongs to.

Example 2. Suppose $\mathit{Proc} = \{p, q\}$. Consider the DTA $\mathcal{D}$ as given by Figure 1. It consists of two timed automata, $\mathcal{A}_p$ and $\mathcal{A}_q$. In both automata, we suppose all states to be final. Moreover, the owner mapping π maps clock x to p and clock y to q. Note that $\mathit{Reset}(\mathcal{A}_p) = \{x\}$ and $\mathit{Reset}(\mathcal{A}_q) = \emptyset$. Before we define the semantics of $\mathcal{D}$ formally and in a slightly more general setting, let us give some intuitions on the behavior of $\mathcal{D}$. If both clocks are completely synchronized, i.e., they follow the same local clock rate, then our model corresponds to a standard network of timed automata [5]. For example, we might execute a within one time unit, and, after one time unit, execute b, ending up in the global state (s_1, r_1) and a clock valuation $\nu(x) = \nu(y) = 1$. If we now wanted to perform a further b, this should happen instantaneously. But this also requires a reset of x in the automaton $\mathcal{A}_p$ and, in particular, a time elapse greater than zero, violating the invariant at the local state r_1. Thus, the word $abab$ will not be in the semantics that we associate with $\mathcal{D}$ wrt. synchronized local-time evolution. Now suppose clock y runs slower than clock x. Then, having executed ab, we might safely execute a further a while resetting x and, then, let some time elapse without violating the invariant. Thus, $abab$ will be contained in the *existential* semantics, as there are local time evolutions that allow for the execution of this word. Observe that a and aa are the only sequences that can be executed no matter what the relative time speeds are: the guard $y \leq 1$ is always satisfied for a while. But we cannot guarantee that the guard $x \geq 1$ and the invariant $y \leq 1$ are satisfied at the same time, which prevents a word containing b from being in the *universal* semantics of $\mathcal{D}$.

The semantics. The semantics of a DTA depends on the (possibly dynamically changing) time rates at the processes. To model this, we assume that these rates depend on some absolute time, i.e., they are given by a tuple $\tau = (\tau_p)_{p \in \mathit{Proc}}$ of functions $\tau_p : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. Thus, each local time function maps every point in global time to some local time instant. Then, we require (justifiably) that these functions are continuous, strictly increasing, and divergent. Further, they satisfy $\tau_p(0) = 0$ for all $p \in \mathit{Proc}$. The set of all these tuples τ is denoted by Rates. We might consider τ as a mapping $\mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0})^{\mathit{Proc}}$ so that, for $t \in \mathbb{R}_{\geq 0}$, $\tau(t)$ denotes the tuple $(\tau_p(t))_{p \in \mathit{Proc}}$.

A distributed system can usually be described with an asynchronous product of automata. Indeed, the semantics of a DTA can be defined with such a product and a mapping that assigns any clock to its owner process: Let $\mathcal{D} = ((\mathcal{A}_p)_{p \in \mathit{Proc}}, \pi)$ with $\mathcal{A}_p = (S_p, \Sigma_p, \mathscr{Z}_p, \delta_p, I_p, \iota_p, F_p)$ be some DTA. We assign to $\mathcal{D}$ the asynchronous product $\mathcal{B}_{\mathcal{D}} = (S, \Sigma, \mathscr{Z}, \delta, I, \iota, F, \pi)$ as one might expect: We set $S = \prod_{p \in \mathit{Proc}} S_p$, $\Sigma = \bigcup_{p \in \mathit{Proc}} \Sigma_p$, $\mathscr{Z} = \bigcup_{p \in \mathit{Proc}} \mathscr{Z}_p$, $\iota = (\iota_p)_{p \in \mathit{Proc}}$, and $F = \prod_{p \in \mathit{Proc}} F_p$. Moreover, for any $s \in S$, we let $I(s) = \bigwedge_{p \in \mathit{Proc}} I_p(s_p)$. Finally, for $s, s' \in S$, $a \in \Sigma_\varepsilon$,

$\varphi \in \mathrm{Form}(\mathcal{Z})$, and $R \subseteq \mathcal{Z}$, we let $(s, a, \varphi, R, s') \in \delta$ if there is $p \in \mathit{Proc}$ such that $(s_p, a, \varphi, R, s'_p) \in \delta_p$ and $s_q = s'_q$ for each $q \in \mathit{Proc} \setminus \{p\}$.

Actually, most variants of a shared-memory model and their semantics can be unified by considering one single state space. This motivates the following definition:

Definition 3. *A* timed automaton with independently evolving clocks (icTA) *over Proc is a tuple $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, I, \iota, F, \pi)$ where $(S, \Sigma, \mathcal{Z}, \delta, I, \iota, F)$ is a timed automaton and $\pi : \mathcal{Z} \to \mathit{Proc}$ maps each clock to a process.*

Thus, the structure $\mathcal{B}_\mathcal{D}$ that we assigned to a DTA $\mathcal{D}$ is an icTA. Most of the following definitions and results are based on this more general notion of a timed system and therefore automatically carry over to the special case of DTAs. We will now define a run of an icTA. Intuitively, this is done in the same spirit as a run of a timed automaton over a timed word except for one difference. The time evolution, though according to absolute time, is perceived by each process as its *local time* evolution. So let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, I, \iota, F, \pi)$ be an icTA. Given a clock valuation ν over $\mathcal{Z}$ and a tuple $t \in \mathbb{R}^{\mathit{Proc}}$, we let the valuation $\nu + t$ be given by $(\nu + t)(x) = \nu(x) + t_{\pi(x)}$ for all $x \in \mathcal{Z}$. For $\tau \in \mathit{Rates}$, a τ-run of $\mathcal{B}$ is a sequence $(s_0, \nu_0) \xrightarrow{a_1, t_1} (s_1, \nu_1) \ldots (s_{n-1}, \nu_{n-1}) \xrightarrow{a_n, t_n} (s_n, \nu_n)$ where $n \geq 0$, $s_i \in S$, $a_i \in \Sigma_\varepsilon$, and $(t_i)_{1 \leq i \leq n}$ is a non-decreasing sequence of values from $\mathbb{R}_{\geq 0}$. Further, $\nu_i : \mathcal{Z} \to \mathbb{R}_{\geq 0}$ with $\nu_0(x) = 0$ for all $x \in \mathcal{Z}$. Finally, for all $i \in \{1, \ldots, n\}$, there are $\varphi_i \in \mathrm{Form}(\mathcal{Z})$ and $R_i \subseteq \mathcal{Z}$ such that the following hold: $(s_{i-1}, a_i, \varphi_i, R_i, s_i) \in \delta$, $\nu_{i-1} + \tau(t') - \tau(t_{i-1}) \models I(s_{i-1})$ for each $t' \in [t_{i-1}, t_i]$, $\nu_{i-1} + \tau(t_i) - \tau(t_{i-1}) \models \varphi_i$, $\nu_i = (\nu_{i-1} + \tau(t_i) - \tau(t_{i-1}))[R_i]$, and $\nu_i \models I(s_i)$. In that case, we write $(\mathcal{B}, \tau) : s_0 \xrightarrow{a_1 \cdot \ldots \cdot a_n} s_n$ or also $(\mathcal{B}, \tau) : s_0 \xrightarrow{a_1 \cdot \ldots \cdot a_i} s_i \xrightarrow{a_{i+1} \cdot \ldots \cdot a_n} s_n$ to abstract from the time instances. The latter thus denotes that $\mathcal{B}$ can, reading w, go from s_0 via s_i to s_n, while respecting the local-time behavior τ.

Definition 4. *Let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, I, \iota, F, \pi)$ be an* icTA *and $\tau \in \mathit{Rates}$. The* language *of $\mathcal{B}$ wrt. τ, denoted by $L(\mathcal{B}, \tau)$, is the set of words $w \in \Sigma^*$ such that $(\mathcal{B}, \tau) : \iota \xrightarrow{w} s$ for some $s \in F$. Moreover, we define $L_\exists(\mathcal{B}) = \bigcup_{\tau \in \mathit{Rates}} L(\mathcal{B}, \tau)$ to be the* existential semantics *and $L_\forall(\mathcal{B}) = \bigcap_{\tau \in \mathit{Rates}} L(\mathcal{B}, \tau)$ to be the* universal semantics *of $\mathcal{B}$.*

If $|\mathit{Proc}| = 1$, then an icTA $\mathcal{B}$ actually reduces to an ordinary timed automaton and we have $L_\forall(\mathcal{B}) = L(\mathcal{B}, \tau) = L_\exists(\mathcal{B})$ for any $\tau \in \mathit{Rates}$. Moreover, if $|\mathit{Proc}| > 1$ and $\tau \in \mathit{Rates}$ exhibits, for all $p \in \mathit{Proc}$, the same local time evolution, then $L(\mathcal{B}, \tau)$ is the language of $\mathcal{B}$ considered as an ordinary timed automaton.

Example 5. A sample icTA $\mathcal{B}$ over set of processes $\{p, q\}$ and $\Sigma = \{a, b, c\}$ is depicted in Figure 2. Assuming $\pi^{-1}(p) = \{x\}$ and $\pi^{-1}(q) = \{y\}$, we have $L_\forall(\mathcal{B}) = \{a, ab\}$, $L(\mathcal{B}, \mathrm{id}) = \{a, ab, b\}$, and $L_\exists(\mathcal{B}) = \{a, ab, b, c\}$ where id_p is the identity on $\mathbb{R}_{\geq 0}$ for all $p \in \mathit{Proc}$ (i.e., id models synchronization of any process with the absolute time).

It is worthwhile to observe that $L(\mathcal{B}, \tau)$ can, in general, have bizarre (non-regular) behavior, if τ is itself a "weird" function. This is one more reason to look at the existential and universal semantics. Let us quantify this with an example. Consider the simple icTA $\mathcal{B}$ over $\mathit{Proc} = \{p, q\}$ fom Figure 3, where $\Sigma = \{a, b\}$, $\pi^{-1}(p) = \{x\}$, and

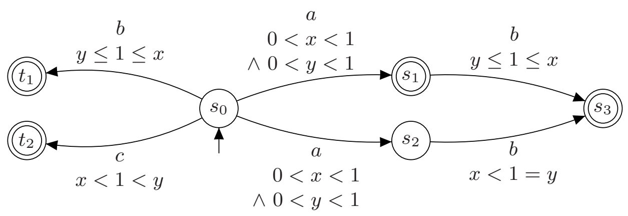


Fig. 2. An icTA $\mathcal{B}$ with independent clocks x and y

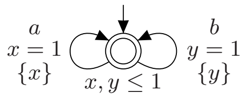


Fig. 3. A "weird" icTA

$\pi^{-1}(q) = \{y\}$. Further, let $\tau = (\mathrm{id}_p, \tau_q)$, where τ_q is any continuous, strictly increasing function such that $\tau_q(0) = 0$ and $\tau_q(n) = 2^n - 0.1$ for any $n \ge 1$. Then, $L(\mathcal{B}, \tau)$ is the set of finite prefixes of the infinite word $bab^2ab^4ab^8ab^{16}a \ldots$, which is not regular.

Finally, the semantics of a DTA is formally described in terms of its icTA.

Definition 6. *For a DTA $\mathcal{D}$ and $\tau \in$ Rates, we set $L(\mathcal{D}, \tau) = L(\mathcal{B}_{\mathcal{D}}, \tau)$ to be the* language *of $\mathcal{D}$ wrt. τ, and we define $L_\exists(\mathcal{D}) = \bigcup_{\tau \in Rates} L(\mathcal{D}, \tau)$ as well as $L_\forall(\mathcal{D}) = \bigcap_{\tau \in Rates} L(\mathcal{D}, \tau)$ to obtain its existential and universal semantics, respectively.*

Example 7. For the DTA $\mathcal{D}$ from Figure 1, we can formalize what we had described intuitively: $L(\mathcal{D}, \mathrm{id}) = \mathit{Pref}(\{aab, aba, baa\})$, $L_\exists(\mathcal{D}) = \mathit{Pref}(\{aab, abab, baab\})$, and $L_\forall(\mathcal{D}) = \mathit{Pref}(\{aa\})$ where, for $L \subseteq \Sigma^*$, $\mathit{Pref}(L) = \{u \mid u, v \in \Sigma^*, u \cdot v \in L\}$.

3 Region Abstraction and the Existential Semantics

Given an icTA $\mathcal{B}$ and a set Bad of undesired behaviors, it is natural to ask if $\mathcal{B}$ is robust against the (unknown) relative clock speeds and faithfully avoids executing action sequences from Bad. This corresponds to checking if $L_\exists(\mathcal{B}) \cap Bad = \emptyset$. In this section, we show that this question is indeed decidable, given that Bad is a regular language. To this aim, we define a partitioning of clock valuations into finitely many equivalence classes and generalize the well-known region construction for timed automata [2].

Let $\mathcal{B} = (S, \Sigma, \mathscr{Z}, \delta, I, \iota, F, \pi)$ be an icTA. For a clock $x \in \mathscr{Z}$, let $C_x \in \mathbb{N}$ be the largest value the clock x is compared with in $\mathcal{B}$ (we assume that such a value exists). We say that two clock valuations ν and ν' over $\mathscr{Z}$ are *equivalent* if the following hold:

- for each $x \in \mathscr{Z}$, $\nu(x) > C_x$ iff $\nu'(x) > C_x$,
- for each $x \in \mathscr{Z}$, $\nu(x) \le C_x$ implies both $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ and ($\mathit{fract}(\nu(x)) = 0$ iff $\mathit{fract}(\nu'(x)) = 0$), and
- for each $p \in \mathit{Proc}$ and $x, y \in \pi^{-1}(p)$ such that $\nu(x) \le C_x$ and $\nu(y) \le C_y$, we have $\mathit{fract}(\nu(x)) \le \mathit{fract}(\nu(y))$ iff $\mathit{fract}(\nu'(x)) \le \mathit{fract}(\nu'(y))$.
  Note that this constraint only applies to clocks that belong to the same process.

An equivalence class of a clock valuation is called a *clock region* (of $\mathcal{B}$). For a valuation ν, $[\nu]$ denotes the clock region that contains ν. The set of clock regions of $\mathcal{B}$ is denoted by $\mathit{Regions}(\mathcal{B})$. Let γ and γ' be two clock regions, say with representatives ν and ν', respectively. We say that γ' is a *accessible* from γ, written $\gamma \preceq \gamma'$, if $\gamma' = \gamma$ or if there is $t \in (\mathbb{R}_{>0})^{\mathit{Proc}}$ such that $\nu' = \nu + t$. Note that $\preceq$ is a partial-order relation.

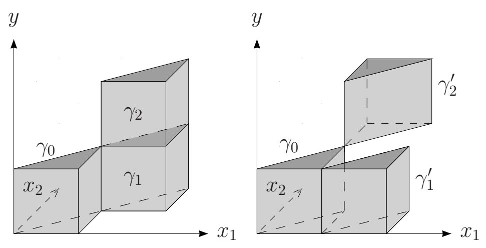


Fig. 4. Accessible and non-accessible regions

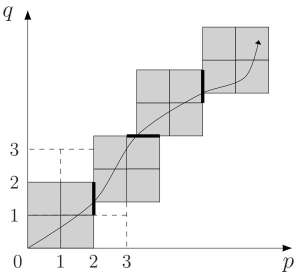


Fig. 5. $dir(\tau) = 010\ldots$

The *successor* relation, written $\gamma \prec\!\!\cdot\ \gamma'$, is as usual defined by $\gamma \prec \gamma'$ and $\gamma'' = \gamma$ or $\gamma'' = \gamma'$ for all clock regions γ'' with $\gamma \preceq \gamma'' \preceq \gamma'$.

Example 8. The accessible-regions relation is illustrated in Figure 4. Suppose we deal with two processes, one owning clocks x_1 and x_2, the other owning a single clock y. Suppose furthermore that, in the icTA at hand, all clocks are compared to the constant 2. Consider the prisms $\gamma_0, \gamma_1, \gamma_2, \gamma_1', \gamma_2'$, each representing a non-border clock region, which are given by the clock constraints $\gamma_0 = (0 < x_2 < x_1 < 1) \wedge (0 < y < 1)$, $\gamma_1' = (0 < x_2 < x_1 - 1 < 1) \wedge (0 < y < 1)$, $\gamma_1 = (1 < x_2 < x_1 < 2) \wedge (0 < y < 1)$, $\gamma_2' = (1 < x_1 < x_2 < 2) \wedge (1 < y < 2)$, and $\gamma_2 = (1 < x_2 < x_1 < 2) \wedge (1 < y < 2)$. We have $\gamma_0 \preceq \gamma_1 \preceq \gamma_2$. However, $\gamma_0 \not\preceq \gamma_1'$ and $\gamma_0 \not\preceq \gamma_2'$.

Let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, I, \iota, F, \pi)$ be an icTA over *Proc*. We associate with $\mathcal{B}$ a non-deterministic finite automaton $\mathcal{R}_\mathcal{B} = (S', \Sigma, \delta', \iota', F')$, called the *region automaton* of $\mathcal{B}$, which is defined as follows: $S' = S \times \mathit{Regions}(\mathcal{B})$, $\iota' = (\iota, [\nu])$ where $\nu(x) = 0$ for all $x \in \mathcal{Z}$, $F' = F \times \mathit{Regions}(\mathcal{B})$, and for $a \in \Sigma_\varepsilon$, $s, s' \in S$, and $\gamma, \gamma' \in \mathit{Regions}(\mathcal{B})$, δ' contains $((s, \gamma), a, (s', \gamma'))$ if

- $a = \varepsilon$, $s = s'$, $\gamma \prec\!\!\cdot\ \gamma'$, and $\nu' \models I(s)$ for some $\nu' \in \gamma'$ (we then call $((s, \gamma), a, (s', \gamma'))$ a *time-elapse transition*), or
- there are $\nu \in \gamma$ and $(s, a, \varphi, R, s') \in \delta$ such that $\nu \models \varphi \wedge I(s)$, $\nu[R] \models I(s')$, and $\nu[R] \in \gamma'$ (we then call $((s, \gamma), a, (s', \gamma'))$ a *discrete transition*).

A part of the region automaton for the icTA from Figure 2 is shown in Figure 10.

Indeed, the language $L(\mathcal{R}_\mathcal{B})$ of the non-deterministic finite automaton $\mathcal{R}_\mathcal{B}$, which is defined as usual, coincides with the existential semantics of $\mathcal{B}$:

Lemma 9. *Let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, I, \iota, F, \pi)$ be an* icTA *and let C be the largest constant a clock is compared with in $\mathcal{B}$. Then, the number of states of $\mathcal{R}_\mathcal{B}$ is bounded by $|S| \cdot (2\,C + 2)^{|\mathcal{Z}|} \cdot |\mathcal{Z}|!$ and we have $L(\mathcal{R}_\mathcal{B}) = L_\exists(\mathcal{B})$.*

Thus, we solved the verification problem stated at the beginning of this section:

Theorem 10. *Model checking* icTAs *wrt. regular negative specifications is decidable.*

4 The Universal Semantics

While the existential semantics allows us to verify negative specifications, the universal semantics is natural when we want to check if our system has some good behavior. By good we mean a behavior that is robust against clock variations. Unfortunately, this problem is undecidable. This is shown for icTAs first and then will be extended to DTAs. Moreover, it turns out to be undecidable if, for a positive specification *Good* containing the behaviors that a system *must* exhibit and an icTA $\mathcal{B}$, we have $\mathit{Good} \subseteq L_\forall(\mathcal{B})$.

Theorem 11. *The following problem is undecidable if $|\mathit{Proc}| \geq 2$: Given an* icTA $\mathcal{B}$ *over Proc, does $L_\forall(\mathcal{B}) \neq \emptyset$ hold?*

Proof. The proof is by reduction from Post's correspondence problem (PCP). An instance *Inst* of the PCP consists of an alphabet A and two morphisms f and g from A^+ to $\{0,1\}^+$. A solution of *Inst* is a word $w \in A^+$ such that $f(w) = g(w)$.

Suppose $\mathit{Proc} = \{p, q\}$ and let $\tau \in \mathit{Rates}$. One may associate with τ two sequences $t\text{-}\mathit{dir}(\tau) = t_1 t_2 \ldots \in (\mathbb{R}_{\geq 0})^\omega$ of time instances and $\mathit{dir}(\tau) = d_1 d_2 \ldots \in \{0,1,2\}^\omega$ of *directions* as follows: for $i \geq 1$, we let first (assuming $t_0 = 0$) $t_i = \min\{t > t_{i-1} \mid \tau_r(t) - \tau_r(t_{i-1}) = 2 \text{ for some } r \in \mathit{Proc}\}$. With this, we set

$$d_i = \begin{cases} 0 & \text{if } \tau_p(t_i) - \tau_p(t_{i-1}) = 2 \text{ and } 1 < \tau_q(t_i) - \tau_q(t_{i-1}) < 2 \\ 1 & \text{if } \tau_q(t_i) - \tau_q(t_{i-1}) = 2 \text{ and } 1 < \tau_p(t_i) - \tau_p(t_{i-1}) < 2 \\ 2 & \text{otherwise} \end{cases}$$

The construction of $\mathit{dir}(\tau)$ is illustrated in Figure 5. The idea is to allow the shape of the relative time-rate function (from τ) to encode a word in $\{0,1,2\}^\omega$. We do this using 2×2-square regions, each consisting of 4 sub-squares as shown. If the rate function leaves this region by the upper boundary or right boundary of the right-upper sub-square, then we write 1 or 0, respectively. If it leaves by any other boundary or by end-points of any sub-square, then we write 2. A new square region is started at the point where the rate function left the old one. Thus, the direction sequences partition the space of time rates.

Roughly speaking, a word is accepted universally by an icTA iff it is accepted for all directions. Our trick will be to define an icTA such that, the PCP instance has a solution w iff the word wb is accepted by the icTA for all directions. Thus, if there is no solution to the PCP, there will be some direction sequence (respectively, local time rates) for which the icTA does not accept.

Let an instance *Inst* of the PCP be given by an alphabet $A = \{a_1, \ldots, a_k\}$ with $k \geq 1$ and two corresponding morphisms f and g. We will construct an icTA $\mathcal{B} = (S, \Sigma, \mathscr{Z}, \delta, I, \iota, F, \pi)$ over the set of processes $\mathit{Proc} = \{p, q\}$ and $\Sigma = \{a_1, \ldots, a_k, b\}$ such that $L_\forall(\mathcal{B}) = \{wb \mid w \in A^+ \text{ and } f(w) = g(w)\}$. First, let $\mathscr{Z} = \{x, y\}$ with $\pi(x) = p$ and $\pi(y) = q$. For $d \in \{0,1,2\}$, we set

$$\mathit{guard}(d) = \begin{cases} x = 2 \ \wedge \ 1 < y < 2 & \text{if } d = 0 \\ y = 2 \ \wedge \ 1 < x < 2 & \text{if } d = 1 \\ ((x \leq 1 \ \vee \ x = 2) \ \wedge y = 2) \ \vee \ (y \leq 1 \ \wedge \ x = 2) & \text{if } d = 2 \end{cases}$$

Moreover, let $\overline{\mathit{guard}}(d) = \bigvee_{d' \in \{0,1,2\} \setminus \{d\}} \mathit{guard}(d')$.

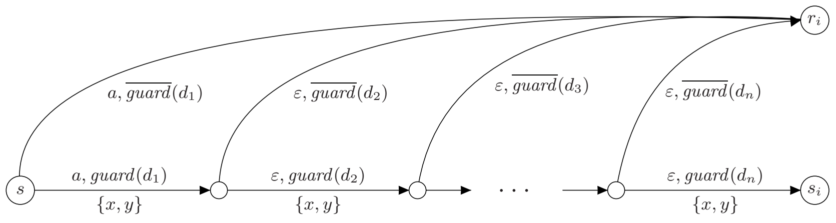

r_i
$a, \overline{guard}(d_1)$
$\varepsilon, \overline{guard}(d_2)$
$\varepsilon, \overline{guard}(d_3)$
$\varepsilon, \overline{guard}(d_n)$
s
$a, guard(d_1)$
$\{x, y\}$
$\varepsilon, guard(d_2)$
$\{x, y\}$
$\dots$
$\varepsilon, guard(d_n)$
$\{x, y\}$
s_i

Fig. 6. Transition macro

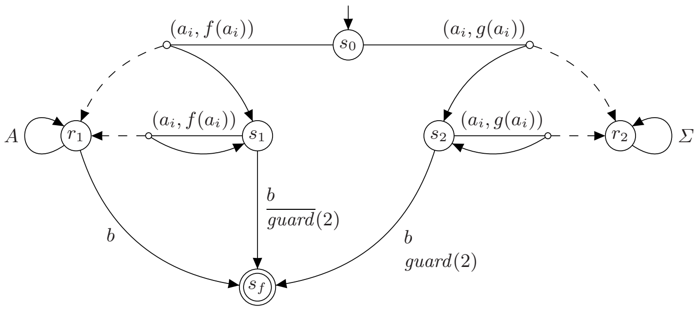
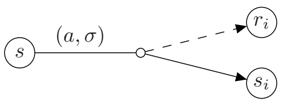


Fig. 7. Encoding of PCP

The final encoding of the given PCP instance in terms of the icTA is given by Figure 7. Hereby, given $a \in A$ and $\sigma = d_1 \dots d_n \in \{0,1,2\}^+$ (with $d_j \in \{0,1,2\}$ for any $j \in \{1,\dots,n\}$), a transition of the form



will actually stand for the sequence of transitions that is depicted in Figure 6, say, with intermediate states $s_{(i,a,\tau,1)}, \dots, s_{(i,a,\tau,n-1)}$.

Example 12. Consider the PCP instance *Inst* given by $A = \{a_1, a_2\}$, $f(a_1) = 101$, $g(a_1) = 1$, $f(a_2) = 1$, $g(a_2) = 01110$ with the obvious solution $w = a_1a_2a_1$. One can check that $a_1a_2a_1b \in L_\forall(\mathcal{B})$. This is illustrated in Figure 8. In the tree depicted, any path corresponds to a finite prefix (of length $|w| + 1$) of some sequence of directions. The edges are labeled by this sequence, where a left-edge is 0, downward is 2 and right-edge is 1. Thus, intuitively, a word wb is in the universal language iff all paths of the tree correspond to accepting runs in $\mathcal{B}$. Now, lets verify that the word wb is accepted by $\mathcal{B}$. If clock rate τ is such that $dir(\tau) \in f(w) \cdot d \cdot \{0,1,2\}^\omega$ with $d \in \{0,1\}$, then the accepting run of $\mathcal{B}$ is the path shown in the left figure, which assigns states s_1 to nodes of the tree and finishes at s_f. If $d = 2$, then the accepting run of $\mathcal{B}$ is the path in the figure on right, which assigns states s_2 appropriately, crucially using the fact that $f(w) = g(w)$, and finally ends at s_f. If the clock rate τ has $dir(\tau)$ different from above

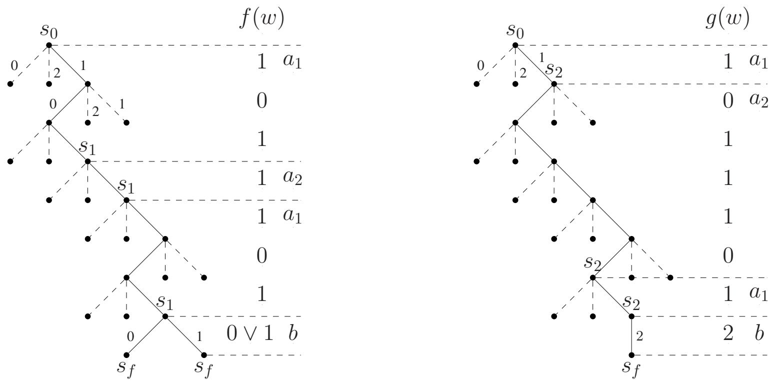


Fig. 8. The tree generated by $w = a_1 a_2 a_1 b$ with respect to f and g

case, it is easy to see that there is an accepting run in which $\mathcal{B}$ reaches state s_f by passing through state r_1.

Let us show that our reduction is indeed correct. In the following, let $\leq$ denote the usual prefix relation on words.

Claim 13. For $\tau \in \mathit{Rates}$ and $w \in A^+$, the following hold:

(1) $f(w) \leq \mathit{dir}(\tau)$ iff $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_1$
(2) $g(w) \leq \mathit{dir}(\tau)$ iff $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_2$
(3) $f(w) \not\leq \mathit{dir}(\tau)$ iff $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} r_1$

With Claim 13, whose proof can be found in the full version [1], we can now show both directions of the correctness of the construction of $\mathcal{B}$.

Let $\tau \in \mathit{Rates}$ and suppose $f(w) = g(w)$. We distinguish three cases: If $\mathit{dir}(\tau) \in f(w) \cdot \{0,1\} \cdot \{0,1,2\}^\omega$, then, by (1), $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_1 \xrightarrow{b} s_f$. If $\mathit{dir}(\tau) \in f(w) \cdot 2 \cdot \{0,1,2\}^\omega$, then $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_2 \xrightarrow{b} s_f$. This follows from (2), since $g(w) = f(w)$. If $f(w) \not\leq \mathit{dir}(\tau)$, then, by (3), $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} r_1 \xrightarrow{b} s_f$. Hence, $wb \in L_\forall(\mathcal{B})$.

Let $w \in A^+$ and suppose $wb \in L_\forall(\mathcal{B})$. Pick $\tau \in \mathit{Rates}$ such that $\mathit{dir}(\tau) \in f(w) \cdot 2 \cdot \{0,1,2\}^\omega$. As $f(w) \leq \mathit{dir}(\tau)$, we have, by (3), $(\mathcal{B}, \tau) : s_0 \overset{w}{\not\rightarrow} r_1$ and $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_1 \overset{b}{\not\rightarrow}$. Thus, we must have $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_2 \xrightarrow{b} s_f$. Hence, by (2), $g(w) \cdot 2 \leq \mathit{dir}(\tau)$. As $f(w), g(w) \in \{0,1\}^*$, we have both $f(w) \cdot 2 \leq \mathit{dir}(\tau)$ and $g(w) \cdot 2 \leq \mathit{dir}(\tau)$, which implies $f(w) = g(w)$. $\square$

Our result can be strengthened and extended to the distributed setting as follows:

Theorem 14. *Suppose $|\mathit{Proc}| \geq 2$. For DTAs $\mathcal{D}$ over Proc, the emptiness of $L_\forall(\mathcal{D})$ is undecidable.*

Proof. We fix $\mathit{Proc} = \{p, q\}$ and the clock distribution $\mathcal{Z}_p = \{x\}$ and $\mathcal{Z}_q = \{y\}$. Each process will be a copy of the automaton that is depicted in Figure 7, except for one

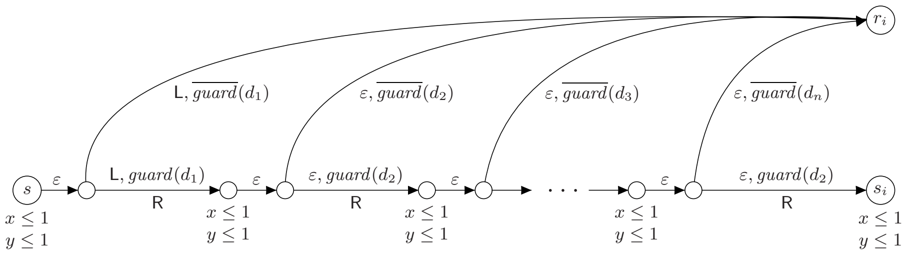

r_i
$\mathsf{L}, \overline{guard}(d_1)$
$\varepsilon, \overline{guard}(d_2)$
$\varepsilon, \overline{guard}(d_3)$
$\varepsilon, \overline{guard}(d_n)$
s
ε
$\mathsf{L}, guard(d_1)$
R
ε
$\varepsilon, guard(d_2)$
R
ε
...
ε
$\varepsilon, guard(d_2)$
R
s_i
$x \leq 1$
$y \leq 1$

Fig. 9. Transition macro for the distributed setting

difference: for process p, the transition macro from Figure 6 is replaced with that from Figure 9 where L is the letter $a \in A$ and R is the singleton set $\{x\}$; for process q, we use the same new macro, but now we have $\mathsf{L} = \varepsilon$ and $\mathsf{R} = \{\mathsf{y}\}$.

To see how this works, we will just point out the difficulties and why the additional states with invariants and ε-transitions fix them. In the transition macro in Figure 6, clocks x and y belonging to different processes are reset at the same time. So, here we have two copies of the same automaton doing the same simulation but reset x in the automaton for process p, and y in the other. But this is not enough, since in the truly distributed setting, we cannot ensure that the clock resets are in sync. This might allow one process to wait while the other has reset its clock and thereby enable (wrong) transitions to state r_i, thus allowing the two automata copies to differ in the simulation. To ensure that the same path is followed, we split each state (except s_i and r_i) into two. The invariant on the first part then ensures that, before the next transition is enabled by the guard (which happens in the second part), both have been reset.

Let us examine this in more detail. Being in two identical copies of a state with an outgoing ε-transition, the ε-transitions might indeed be taken asynchronously by p and q. However, the following transitions will be performed synchronously by both processes. Assume first that p follows a transition of the form $(s_p, a, guard(d), \{x\}, s'_p)$ before process q moves. As $guard(d)$, where $d \in \{0, 1\}$, is satisfied when p goes to s'_p, the value of both clocks exceeds 1. But as x is reset at the same time whereas y is not, the invariant associated with s'_p is violated, which is a contradiction. Thus, q has to take the corresponding transition, which is of the form $(s_q, a, guard(d), \{y\}, s'_q)$, simultaneously. This explains why we use 2×2-squares as in Figure 5 and corresponding guards. In DTAs, they allow us to check when one clock has been reset and other has not. Now consider the case where p performs a transition of the form $(s_p, a, \overline{guard}(d), \emptyset, s'_p)$. When p executes its transition, at least one clock has reached the value 2. As this clock cannot be reset anymore, q is obliged to follow instantaneously the corresponding transition of the form $(s_q, a, \overline{guard}(d), \emptyset, s'_q)$, to reach a final state. □

Along the lines of the proofs of Theorems 11 and 14, we can show the following theorem, from which we derive the subsequent negative result (see [1] for details):

Theorem 15. *Suppose that $|Proc| \geq 2$. For DTAs $\mathcal{D}$ over Proc, it is undecidable if $L_\forall(\mathcal{D}) = \Sigma^*$ (where Σ is the set of actions of $\mathcal{B}_\mathcal{D}$).*

Theorem 16. *Model checking DTAs over at least two processes against regular positive specifications is undecidable.*

5 Playing with Local Time Rates

We have shown that it is undecidable to check if there is some word that is accepted under all clock rates by a given icTA $\mathcal{B}$. It is natural to ask if it is possible to restrict the independence of local time rates in some way to get decidability. For instance, we could insist that the ratio or the difference of local times in different processes must always be bounded. Unfortunately, this does not help. In fact, it turns out that our proof in Section 4 can be used to show that both these restrictions are already undecidable.

Let us formalize this. We will restrict to two processes, $\mathit{Proc} = \{p, q\}$. We note however that the following definitions can be easily generalized to more processes. For a rational number $k \geq 1$, we define $\mathit{Rates}_{\text{rat}}(k) = \{\tau = (\tau_p, \tau_q) \in \mathit{Rates} \mid \frac{1}{k} \leq \frac{\tau_p(t)}{\tau_q(t)} \leq k$ for all $t \in \mathbb{R}_{>0}\}$. This is the set of all rate-function tuples such that the ratio of the local times in the two processes are always bounded by fixed rationals. Further, for a rational number $\ell \geq 0$, $\mathit{Rates}_{\text{dif}}(\ell) = \{\tau = (\tau_p, \tau_q) \in \mathit{Rates} \mid |\tau_p(t) - \tau_q(t)| \leq \ell$ for all $t \in \mathbb{R}_{\geq 0}\}$. These are the rate function tuples for which the difference between the local times in the two processes are bounded by some constant. Accordingly, for an icTA or a DTA $\mathcal{B}$, we define $L_{\forall}^{\text{rat},k}(\mathcal{B}) = \bigcap_{\tau \in \mathit{Rates}_{\text{rat}}(k)} L(\mathcal{B}, \tau)$ and $L_{\forall}^{\text{dif},\ell}(\mathcal{B}) = \bigcap_{\tau \in \mathit{Rates}_{\text{dif}}(\ell)} L(\mathcal{B}, \tau)$.

Theorem 17. *For* icTAs *or DTAs $\mathcal{B}$ over $\mathit{Proc} = \{p, q\}$,*

1. *the emptiness of $L_{\forall}^{\text{rat},1}(\mathcal{B}) = L_{\forall}^{\text{dif},0}(\mathcal{B})$ is decidable.*
2. *the emptiness of $L_{\forall}^{\text{rat},k}(\mathcal{B})$ is undecidable for every rational $k > 1$.*
3. *the emptiness of $L_{\forall}^{\text{dif},\ell}(\mathcal{B})$ is undecidable for every rational $\ell > 0$.*

To prove the theorem, we need the following lemma.

Lemma 18. *Let $k > 1$, $\ell > 0$ be some fixed rationals. For all $\sigma \in \{0, 1, 2\}^*$, there exists $\tau \in \mathit{Rates}_{\text{rat}}(k) \cap \mathit{Rates}_{\text{dif}}(\ell)$ such that σ is a prefix of $\mathit{dir}(\tau)$.*

Proof. Let $\sigma = d_1 d_2 \ldots d_n \in \{0, 1, 2\}^*$ be of length n. We define τ (in terms of $n+1$ points) as follows: τ_p is the piecewise linear function with $\tau_p(2i) = x_i$ for $i \in \{0, \ldots, n\}$ and $\tau_p(2n + t) = x_n + t$ for all $t \in \mathbb{R}_{\geq 0}$. Similarly, τ_q is defined as the piecewise linear function with $\tau_q(2i) = y_i$ for $i = \{0, \ldots, n\}$ and $\tau_q(2n + t) = y_n + t$ for $t \in \mathbb{R}_{\geq 0}$. The points (x_i, y_i) are defined by $x_0 = y_0 = 0$ and, for $i \in \{1, \ldots, n\}$, $x_i = 2i - \alpha |d_1 \ldots d_i|_1$ and $y_i = 2i - \alpha |d_1 \ldots d_i|_0$ ($|\sigma'|_d$ denoting the number of occurrences of d in σ'), where α is a rational parameter to be fixed.

With the above definition, we observe that, for all i, we have $|x_i - y_i| \leq i\alpha$, and, for $i > 0$, we have $1 - \frac{\alpha}{2} \leq \frac{x_i}{y_i} \leq \frac{1}{1 - \alpha/2}$. Thus, by choosing $\alpha = \min\{\frac{\ell}{n}, 2(1 - \frac{1}{k})\}$, we can check that $\tau \in \mathit{Rates}_{\text{rat}}(k) \cap \mathit{Rates}_{\text{dif}}(\ell)$. Also it is easy to see that $\mathit{dir}(\tau) = \sigma \cdot 2^{\omega}$, which proves the lemma. □

Now, we can prove Theorem 17. For $k = 1$ or $\ell = 0$, the sets $Rates_{\text{rat}}(k)$ and $Rates_{\text{dif}}(\ell)$ consist of exactly the tuples in which time evolves at the same rate in both processes. Thus the sets are identical and correspond to an ordinary timed automaton so that emptiness is decidable.

Now, let $k > 1$ and $\ell > 0$. Given a PCP instance as before, we again consider the icTA (or DTA) $\mathcal{B}$ from Section 4. We want to show that $w \in A^+$ is solution iff $wb \in L_\forall(\mathcal{B}) = L_\forall^{\text{rat},k}(\mathcal{B}) = L_\forall^{\text{dif},\ell}(\mathcal{B})$. One direction is trivial. If, for $w \in A^+$, we have $f(w) = g(w)$, then $wb \in L_\forall(\mathcal{B})$, and this implies that $wb \in L_\forall^{\text{rat},k}(\mathcal{B})$ and $wb \in L_\forall^{\text{dif},\ell}(\mathcal{B})$. On the other hand, if $wb \in L_\forall^{\text{rat},k}(\mathcal{B})$ or $wb \in L_\forall^{\text{dif},\ell}(\mathcal{B})$, then, by Lemma 18, we pick $\tau \in Rates_{\text{rat}}(k) \cap Rates_{\text{dif}}(\ell)$ such that $dir(\tau) = f(w) \cdot 2 \cdot 2^\omega$, and the remaining part of the proof follows as before.

6 The Reactive Semantics

The universal semantics described in the previous section is a possible way to implement positive specifications, i.e, to make sure that our system must satisfy some behavior irrespective of the time/clock evolution. Unfortunately, since emptiness is undecidable even for bounded restrictions, it is not of any practical use. We would indeed like a semantics that describes only regular behaviors.

There is another subtle point for looking for other semantics. When we want to check if the system satisfies a positive specification, we would like to be able to design a controller which can actually do this. For this, the semantics has to be "reactive" in some sense. The universal semantics fails in this, in the sense that, to choose a correct run in the system, we might need to know the future time rates.

In this section, we introduce a new game-like semantics that solves both the above mentioned worries. It is regular and it is "reactive". Formally, we will describe it using an alternating automaton, which is based on the region automaton introduced in Section 3. Intuitively, *time-elapse* transitions are controlled by the environment whereas *discrete* transitions are controlled by the system that aims at exhibiting some behavior. This *game* is not *turn-based* because the system should be able to execute several discrete transitions while staying in the same region. After moving from some region to a successor region, the environment hands over the control to the system so that the system always has a chance to execute some discrete transition. On the other hand, after executing some discrete transition, the system may either keep the control or hand it over to the environment.

As suggested, our reactive semantics will be described by alternating automata. Since icTAs or DTAs have ε-transitions, we define an *alternating automaton with ε-transitions* (ε-AA) as a tuple $\mathcal{A} = (S, \Sigma, \delta, \iota, F)$ where S is a finite set of *states*, $\iota \in S$ is the *initial state*, $F \subseteq S$ is the set of *final states*, and $\delta : S \times \Sigma_\varepsilon \to \mathbb{B}^+(S)$ is the *alternating transition function*. Here, $\mathbb{B}^+(S)$ denotes positive boolean combinations of states from S.

As usual, a run of an ε-AA will be a (doubly) labeled finite tree. We assume the reader to be familiar with the notion of trees and only mention that we deal with structures (V, σ, μ) where V is the finite set of nodes with a distinguished root, and both σ and μ are node-labeling functions. Given a node $u \in V$, the set of children of u is denoted children(u). Let $w = a_1 \dots a_{|w|} \in \Sigma^*$ be a finite word. A run of $\mathcal{A}$ on w is a doubly labeled finite tree $\rho = (V, \sigma, \mu)$ where $\sigma : V \to S$ is the *state-labeling* function and $\mu : V \to \{0, \dots, |w|\}$ is the *position-labeling* function such that, for each node $u \in V$, the following hold:

- if u is the root, then $\sigma(u) = \iota$ and $\mu(u) = 0$ (we start in the initial state at the beginning of the word),
- if u is not a leaf (i.e., children$(u) \neq \emptyset$), then we have
  - either $\mu(u') = \mu(u)$ for all $u' \in \text{children}(u)$ and in this case $\{\sigma(u') \mid u' \in \text{children}(u)\} \models \delta(\sigma(u), \varepsilon)$
  - or $\mu(u') = \mu(u) + 1 = i \leq n$ for all $u' \in \text{children}(u)$ and in this case $\{\sigma(u') \mid u' \in \text{children}(u)\} \models \delta(\sigma(u), a_i)$.

The run is accepting if all leaves are labeled with $F \times \{|w|\}$. The set of words from Σ^* that come with an accepting run is denoted by $L(\mathcal{A})$.

Lemma 19 (cf. [4]). *Given an ε-AA $\mathcal{A}$ with n states, one can construct a non-deterministic finite automaton with $2^{\mathcal{O}(n^2)}$ states that recognizes $L(\mathcal{A})$.*

Let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, I, \iota, F, \pi)$ be an icTA over *Proc*. We associate with $\mathcal{B}$ an ε-AA $\mathcal{A}_\mathcal{B} = (S', \Sigma, \delta', \iota', F')$ as follows: First, let $S' = S \times \mathit{Regions}(\mathcal{B}) \times \{0, 1\}$. Intuitively, tag 0 is for *system positions* while tag 1 is for *environment positions* (recall that the environment controls how time elapses whereas the system wants to accept some word). Then, $\iota' = (\iota, [\nu], 0)$ where $\nu(x) = 0$ for each $x \in \mathcal{Z}$, and $F' = F \times \mathit{Regions}(\mathcal{B}) \times \{0, 1\}$. Finally, for $(s, \gamma) \in S \times \mathit{Regions}(\mathcal{B})$ and $a \in \Sigma_\varepsilon$, we let

$$\delta'((s,\gamma,1),a) = \text{False} \quad \text{if } a \neq \varepsilon \qquad \delta'((s,\gamma,1),\varepsilon) = \bigwedge\{(s,\gamma',0) \mid \gamma \prec \gamma'\}$$

$$\delta'((s,\gamma,0),a) = \begin{cases} \bigvee\{(s',\gamma',0) \mid (s,\gamma) \xrightarrow{a}_{\mathrm{d}} (s',\gamma')\} & \text{if } a \neq \varepsilon \text{ or } \gamma \text{ maximal} \\ (s,\gamma,1) \vee \bigvee\{(s',\gamma',0) \mid (s,\gamma) \xrightarrow{\varepsilon}_{\mathrm{d}} (s',\gamma')\} & \text{otherwise} \end{cases}$$

where $\xrightarrow{a|\varepsilon}_{\mathrm{d}}$ denotes a discrete transition of the region automaton $\mathcal{R}_\mathcal{B}$ (Section 3).

Definition 20. *For an* icTA *$\mathcal{B}$, let $L_{react}(\mathcal{B}) = L(\mathcal{A}_\mathcal{B})$ be the* reactive semantics *of $\mathcal{B}$. Moreover, for a DTA $\mathcal{D}$, $L_{react}(\mathcal{D}) = L_{react}(\mathcal{B}_\mathcal{D})$ is the* reactive semantics *of $\mathcal{D}$.*

Example 21. Consider the icTA $\mathcal{B}$ from Figure 2. A part of its ε-AA $\mathcal{A}_\mathcal{B}$ is shown in Figure 10. States with tag 0 are depicted as ovals and are existential (non-deterministic) states and states with tag 1 are depicted as rectangles and are universal states. We have, e.g., $\delta'(r_1, \varepsilon) = r_3 \wedge r_4 \wedge r_5$. Note, however, that a transition from an oval to a rectangles should actually be split into two transitions, which is omitted in the picture. For example, there is a state r_1' between r_0 and r_1 which resembles r_1 but is tagged 0. Similarly, there is another state r_2' between r_0 and r_2, and we have $\delta'(r_0, a) = r_1' \vee r_2'$.

The following theorem follows from Lemma 19:

Theorem 22. *Let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, I, \iota, F, \pi)$ be an* icTA *and let n be the number of states of $\mathcal{R}_\mathcal{B}$ (which is bounded by $|S| \cdot (2\,C+2)^{|\mathcal{Z}|} \cdot |\mathcal{Z}|!$ where C is the largest constant a clock is compared with in $\mathcal{B}$). Then, $L_{react}(\mathcal{B})$ is regular and one can compute a non-deterministic finite automaton with $2^{\mathcal{O}(n^2)}$ states that recognizes $L_{react}(\mathcal{B})$.*

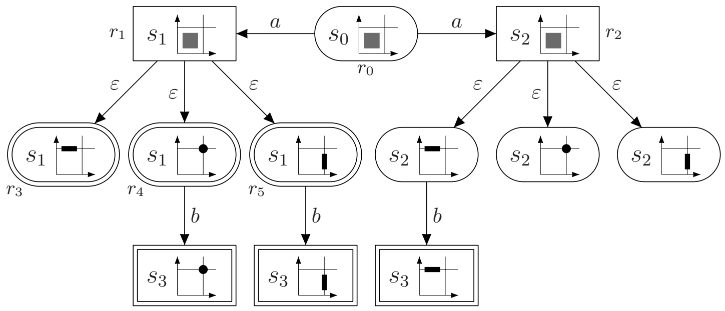


Fig. 10. Part of the region/alternating automaton for the icTA from Figure 2

The following inclusion property, whose proof can be found in [1], allows us to check an icTA for positive specifications. The subsequent proposition then establishes that inclusion actually forms a strict hierarchy of our semantics.

Proposition 23. *For any* icTA $\mathcal{B}$, $L_{react}(\mathcal{B}) \subseteq L_{\forall}(\mathcal{B})$.

Proposition 24. *Suppose that* $|Proc| \geq 2$. *There are some DTA* $\mathcal{D}$ *over Proc and some* $\tau \in Rates$ *such that* $L_{react}(\mathcal{D}) \subsetneq L_{\forall}(\mathcal{D}) \subsetneq L(\mathcal{D}, \tau) \subsetneq L_{\exists}(\mathcal{D})$.

Proof. Consider the icTA $\mathcal{B}$ from Figure 2. Recall that $L_{react}(\mathcal{B}) = \{a\}$, $L_{\forall}(\mathcal{B}) = \{a, ab\}$, $L(\mathcal{B}, \mathrm{id}) = \{a, ab, b\}$, and $L_{\exists}(\mathcal{B}) = \{a, ab, b, c\}$. As $\mathcal{B}$ does not employ any reset, we may view it as a DTA where $\mathcal{B}$ models a process owning clock x, and where a second process, owning clock y, does nothing, but is in a local accepting state. □

7 Future Work

We plan to investigate the expressive power of DTAs and, in particular, the *synthesis problem*: For which (global) specifications *Spec* can we generate a DTA $\mathcal{D}$ (over some given system architecture) such that $L_{react}(\mathcal{D}) = Spec$? A similar synthesis problem has been studied in [8] in the framework of untimed distributed channel systems. There, additional messages are employed to achieve a given global behavior. In this context, it would be favorable to have partial-order based specification languages and a partial-order semantics for DTAs (see, for example, [11]).

References

1. Akshay, S., Bollig, B., Gastin, P., Mukund, M., Narayan Kumar, K.: Distributed timed automata with independently evolving clocks. Research Report LSV-08-19, ENS Cachan (2008)
2. Alur, R., Dill, D.L.: A theory of timed automata. TCS 126(2), 183–235 (1994)
3. Bengtsson, J., Jonsson, B., Lilius, J., Yi, W.: Partial order reductions for timed systems. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 485–500. Springer, Heidelberg (1998)

4. Birget, J.-C.: State-complexity of finite-state devices, state compressibility and incompressibility. Mathematical Systems Theory 26(3), 237–269 (1993)
5. Bouyer, P., Haddad, S., Reynier, P.-A.: Timed unfoldings for networks of timed automata. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218. Springer, Heidelberg (2006)
6. De Wulf, M., Doyen, L., Markey, N., Raskin, J.-F.: Robustness and implementability of timed automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 118–133. Springer, Heidelberg (2004)
7. Dima, C., Lanotte, R.: Distributed time-asynchronous automata. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711. Springer, Heidelberg (2007)
8. Genest, B.: On implementation of global concurrent systems with local asynchronous controllers. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 443–457. Springer, Heidelberg (2005)
9. Henzinger, T.A.: The theory of hybrid automata. In: Proc. of LICS 1996 (1996)
10. Larsen, K.G., Pettersson, P., Yi, W.: Compositional and symbolic model-checking of real-time systems. In: Proc. of RTSS 1995, p. 76. IEEE Computer Society, Los Alamitos (1995)
11. Lugiez, D., Niebert, P., Zennou, S.: A partial order semantics approach to the clock explosion problem of timed automata. TCS 345(1), 27–59 (2005)
12. Puri, A.: Dynamical properties of timed automata. Discrete Event Dynamic Systems 10(1-2), 87–113 (2000)