Timed languages, dyadic approximations and regular tree grammars

Jean-François Antoniotti and Dominique Luzeaux
Lab. Perception pour la RObotique CTA/GIP,
16bis, av. Prieur de la Côte d'Or, 94114 Arcueil Cedex, FRANCE
{antoniot,luzeaux}@etca.fr

Introduction

Our research on the modelization of the loop action-perception for an autonomous robot has led us naturally towards a tree-like representation of that interaction: a node is labeled by a **perceptive context** of the robot, the different branches correspond to the various available actions and the children nodes are then the effect of these different actions on the original context.

This tree-like view allows us to state rigorously the problems of **viable control** in terms of **games**: the existence of a controller is reduced to the existence of strategies which allow to walk through the trees representing the system in order to remain within a given class of contexts (viability goal). Such game approaches are widely used in computer science (BL69; Mos89; NYY92), and appear in hybrid control theory too (TLS98).

Furthermore it is necessary to describe these tree sets as simply as possible, in order to **decide** for the existence of a controller, and in the best case to **synthesize** such a controller (Tho95). In this paper, we study **timed automata** from the point of view of **regular tree grammars** showing how our approach can be applied to this central class of decidable hybrid systems.

Timed automata and duration semantic

Our model of timed automaton differs from the one proposed in (AD94). Our choice follows (ACM97) where a duration semantic is defined; we believe that the ideas of this paper are more understandable in this framework and that these two models are similar enough to ensure the generality of our approach.

Let us recall the notations and definitions from (ACM97).

Notation: An **interval with integral boundaries** is an interval (l, u) (boundaries may be open or closed) where $l \in \mathbb{N}$, $u \in \mathbb{N} \cup \{\infty\}$ (the case $\infty]$ is not considered) and $l \leq u$.

Definition 1 (Timed automata) *A* **timed automaton** $\mathcal{A} = (Q, C, \Delta, \Sigma, \lambda, S, F)$ *is a 7-tuple where Q is a finite set of* states, *C a finite set of* clocks, *Σ an* output alphabet, *Δ a* transition relation, *λ an* output function, *$S \subset Q$ a set of* initial *states and $F \subset Q$ a set of* final *states. An element of the transition relation is (q, ϕ, ρ, q') where q and q' are states, $\rho \subset C$ and ϕ – the* **transition guard** *– is a boolean combination of formulae $c \in I$ for a clock c and an interval with integral boundaries I.*

A **clock valuation** is a function $v : C \to \mathbb{R}^+$ assigning to each clock a positive value. The reset function for a set $\rho \subset C$ maps v to the valuation defined by $Reset_\rho(v)(c) = \begin{cases} 0 & \text{if } c \in \rho, \\ c & \text{otherwise.} \end{cases}$

A **finite run of length** n of the timed automaton is a sequence

$$\underset{t_0}{(q_0, v_0)} \xrightarrow{\delta_1} \underset{t_1}{(q_1, v_1)} \xrightarrow{\delta_2} \dots \xrightarrow{\delta_n} \underset{t_n}{(q_n, v_n)},$$

with $\delta_i \in \Delta$ and $t_i \in \mathbb{R}^+$, which satisfies both following conditions:

Time progression: $t_0 < t_1 < t_2 < \dots < t_n$;

Succession: If $\delta_i = (q, \phi, \rho, q')$ then $q_{i-1} = q$, $q_i = q'$, the valuation $v_{i-1} + (t_i - t_{i-1})$ – addition of a function and a scalar – satisfies the transition guard ϕ and $v_i = Reset_\rho(v_{i-1} + (t_i - t_{i-1}))$.

An **accepting** run should satisfy the following supplementary conditions:

Initialization: $q_0 \in S$, $t_0 = 0$ and $v_0 = 0_{\mathcal{F}}$ (zero function),

Termination: $q_n \in F$.

Such an automaton emits a signal $\sigma \in \Sigma$ from each state and is able to move from a state to another when its clocks satisfy the guard of the transition to be fired.

One can consider such a automaton modeling a **real-time system** whose behavior is:

- emiting $\lambda(q)$, the system signals that its current internal state is q;
- if the clocks have the right values (depending on the current state q) then it must move to state q' (according with the transition relation);
- it has to make a move before the clocks exceed the right values.

This **duration semantic** (emit $\lambda(q)$ while staying in state q) is now formally defined.

Definition 2 *A* **signal** *of length $k > 0$ is a right-continuous function $\xi : [0, k[\rightarrow \Sigma$ with a finite number of discontinuities. The length of ξ is denoted by $|\xi|$.*

Remark: Such a function can be put under the equivalent form $\xi = {\sigma_1}^{r_1}{\sigma_2}^{r_2} \ldots {\sigma_n}^{r_n}$ with the $\sigma_i \in \Sigma$ and the r_i strictly positive for all $0 < i \leq n$, and $\sum_{i=1}^{n} r_i = k$. The logical part of the signal – denoted by *untimed*(ξ) – is the word $\sigma_1\sigma_2 \ldots \sigma_n$.

Definition 3 *If one notes $S(\Sigma)$ the set of signals whose codomain is Σ, the* **renaming** *g_* from $S(\Sigma_1)$ to $S(\Sigma_2)$ is the natural homorphism induced by the letter-to-letter function $g : \Sigma_1 \rightarrow \Sigma_2$.*

Definition 4 *Given two signals ξ and ξ' of length k and k', their* **concatenation** *is the signal denoted by $\xi \circ \xi'$, of length $k + k'$, defined by:*

$$\xi \circ \xi'(t) = \begin{cases} \xi(t) & \text{if } t < k, \\ \xi'(t-k) & \text{if } k \leq t < k + k'. \end{cases}$$

Definition 5 (Duration semantics) *The* **trace** *of a run of length n is the signal of length $t_n - t_0$ defined by:*

$$\lambda(q_0)^{t_1-t_0}\lambda(q_1)^{t_2-t_1} \ldots \lambda(q_{n-1})^{t_n-t_{n-1}}.$$

The **language** *of the timed automaton $\mathcal{A}$, $L(\mathcal{A})$, is the set of all the traces of the accepting runs.*

Remark: q_n is considered just as an ending state of the run, and $\lambda(q_n)$ does not appear in this definition.

From a timed automaton to several one-clock automata

In this section, we give a list of properties satisfied by timed automata which will be useful for the coming proofs.

- A state is **resetting** if all its incoming transitions reset exactly the same set of clocks. An automaton is **state-reset** if all its states are resetting.
- An automaton is **self-loop free** if it does not contain transitions (q, ϕ, ρ, q).
- An automaton is **conjunctive** if all its transition guards are conjunctions of simple tests $c \in I$ or $\neg(c \in I)$.
- An automaton is **state-output** if $\Sigma = Q$ and $\lambda = Id$. Given an automaton $\mathcal{A} = (Q, C, \Delta, \Sigma, \lambda, S, F)$ one can give a state-output automaton $\mathcal{A}' = (Q, C, \Delta, \Sigma, Id, S, F)$ which is equivalent to $\mathcal{A}$ up to the renaming λ_*. The state-output property allows us **to read** into signals the underlying runs of the automaton.

In (ACM97), a procedure is given which transforms a timed automaton into a state-reset self-loop free conjunctive and state-output automaton. A proof of the following lemma can be found there. It shows the interest of such a transformation which transfers constructively results obtained with one-clock automata to automata with several clocks.

Lemma 1 *Let $\mathcal{A}$ be a timed automaton with k clocks. We can construct k one-clock automata $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ and a renaming φ such that $L(\mathcal{A}) = \varphi(\bigcap_{i=1}^{k} L(\mathcal{A}_i))$.*

Roughly speaking, first we transform the automaton into a state-reset, self-loop and conjunctive one by splitting the states. In this first step, states keep their output letters. Then, we consider the underlying state-output automaton which is equivalent to the former up to renaming, as mentioned earlier. Each signal of the language of this automaton is the trace of exactly one run, and this run is the same in all sub-automata $\mathcal{A}_i$ where we only consider clock c_i to evaluate the transitions guards, forgetting the value of clocks c_j for $j \neq i$. Note that automata $\mathcal{A}_i$ are trivially state-reset, self-loop, conjunctive and state-output.

From now on, and unless stated otherwise, we consider only "transformed" timed automata, i.e. holding the state-reset, self-loop, conjunctive and state-output properties.

Signals and trees : the basic representation

Motivation

In (ACM97), Maler et al. prove that a timed language can be described by a **timed** regular expression built from the classical operators (HU79) with the additional operators $\langle\rangle_I$ where I is an interval with integer boundaries. Such an operator allows the "jump" between a word $w \in \Sigma^+$ and the set of signals, the length of which belongs to I and the logical part is w.

This result is an extension of Kleene's theorem *[ibid.]* for arbitrary timed automata, and the main reason is because: if $0 = \tau_0 < \tau_1 < \ldots < \tau_n < \tau_{n+1} = \infty$ is the ordered list of the constants appearing in its transition guards, a timed automaton behaves in a **regular** manner on the intervals $]\tau_i, \tau_{i+1}[$ for $0 \leq i \leq n$: on such an interval, truth values of transition guards are constant, and the timed automaton behaves like the correspondent classical sub-automaton. Such intervals being in finite number for a given automaton, it is possible to describe uniformly the set of recognized signals as the concatenation of signals $\langle r \rangle_I$ where r is a classical regular expression[1].

This fundamental property being identified, our problem is now: does there exist other "classical" syntactic representations for timed languages? And is it possible to find a family of such expressions closed under concatenation?

The goal of the following paragraphs is to prove that **tree regular expressions** are a good candidate, and to discuss about the implications of this fact in term of control for timed automata.

[1]This is particularly clear if one considers a one-clock automaton which is never reset, i.e. such that its clock's value corresponds to the duration elapsed since the start of the run.

Dyadic signals

Our goal being to describe syntactically signals accepted by a timed automaton, we are facing off the classical representation problem of real-valued durations. To overcome this problem, we introduce the notion of dyadic signal ; durations defining such signals are multiples of a sample step equal to 2^{-e} for an integer $e \in \mathbb{N}$.

Definition 6 *A* **dyadic signal of length** 1 *is a signal of the form:*

$$\xi = {\sigma_1}^{p_1.2^{-e}}{\sigma_2}^{p_2.2^{-e}}\dots{\sigma_n}^{p_n.2^{-e}}$$

where e is an integer and p_i are strictly positive integers s.t. $\sum_{i=1}^{n} p_i \cdot 2^{-e} = 1$. A **dyadic signal** *(of length k) is the homothetic image of a 1-length dyadic signal: $\xi = k \cdot ({\sigma_1}^{p_1.2^{-e}}{\sigma_2}^{p_2.2^{-e}}\dots{\sigma_n}^{p_n.2^{-e}})$.*

In this section, we show how to approximate a signal with a dyadic signal of the same length. For this purpose, we define a projection from 1-length signals to 1-length dyadic signals.

The projection follows two steps :

- We isolate discontinuity points of the signal, foreclosing them within dyadic boundary intervals as small as needed. Outside these intervals, the value of the projected signal is given without ambiguities by the original one.
- We "fill in the holes", deciding for the value inside the intervals by watching the value either at the left of the interval (**low approximation**) or at the right (**high approximation**).

The following lemma corresponds to the high approximation case.

Lemma 2 *Let $\xi = {\sigma_1}^{r_1}{\sigma_2}^{r_2}\dots{\sigma_n}^{r_n}$ be a 1-length signal. Letting:*

- $t_i = \sum_{j=1}^{i} r_j$ *for* $0 < i \leq n$,
- $e = \min\{e : 2^{-e} \leq \min\{r_i\}_{0<i\leq n}\}$ *and* $p_i = \max\{p : p \cdot 2^{-e} \leq r_i\}$ *for* $0 < i \leq n$,
- $p'_1 = p_1$ *and, for* $0 < i < n$, $p'_{i+1} = p_{i+1}$ *if* $r_{i+1} - p_{i+1} \cdot 2^{-e} + t_i - \sum_{j=1}^{i} p'_j \cdot 2^{-e} < 2^{-e}$, *and* $p'_{i+1} = p_{i+1} + 1$ *otherwise,*

we get:

$$\sum_{j=1}^{i} p'_j = \max\{p' : p' \cdot 2^{-e} \leq t_i\} \text{ for all } 0 < i \leq n.$$

The differents objects of this lemma are illustrated by figure 1.

PROOF: First note that e is well defined because the r_i are strictly positive, and the set $\{e : 2^{-e} \leq \min\{r_i\}_{0<i\leq n}\}$ is nonempty. Moreover, the p_i are strictly positive : $1 \in \{p : p \cdot 2^{-e} \leq r_i\}$ for all $0 < i \leq n$, by definition of e.

By induction on i we show that for all $0 < i \leq n$, there exists $0 \leq \epsilon_i < 2^{-e}$ s.t. $t_i = \sum_{j=1}^{i} p'_j \cdot 2^{-e} + \epsilon_i$.

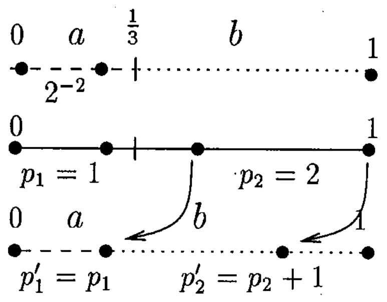

Figure 1: High approximation of the signal $\xi = a^{\frac{1}{3}}b^{\frac{2}{3}}$.

The case $i = 1$ follows from definition of p'_1. Suppose the property true at rank i. Writing $r_{i+1} = p_{i+1} \cdot 2^{-e} + \epsilon_2$ with $0 \leq \epsilon_2 < 2^{-e}$ (such an ϵ_2 exists by construction of p_{i+1}) and $r_{i+1} = t_{i+1} - \sum_{j=1}^{i} p'_j \cdot 2^{-e} + \sum_{j=1}^{i} p'_j \cdot 2^{-e} - t_i$, we get the equality:

$$(\star)\ t_{i+1} - \sum_{j=1}^{i} p'_j \cdot 2^{-e} - p_{i+1} \cdot 2^{-e} = \epsilon_i + \epsilon_2.$$

The induction step follows by considering $(\star)$ and the following two cases. If $0 \leq \epsilon_i + \epsilon_2 < 2^{-e}$ then $p'_{i+1} = p_{i+1}$ and we let $\epsilon_{i+1} = \epsilon_i + \epsilon_2$; if $2^{-e} \leq \epsilon_i + \epsilon_2 < 2 \cdot 2^{-e}$ then $p'_{i+1} = p_{i+1} + 1$ and we let $\epsilon_{i+1} = \epsilon_i + \epsilon_2 - 2^{-e}$. □

Remark: $t_n = 1$, so the equality $t_n = \sum_{i=1}^{n} p'_i \cdot 2^{-e} + \epsilon_n$ with $0 \leq \epsilon_n < 2^{-e}$ implies $\sum_{i=1}^{n} p'_i = 2^e - \epsilon_n \cdot 2^e$ with $0 \leq \epsilon_n \cdot 2^e < 1$; $\sum_{i=1}^{n} p'_i$ being an integer, this finally implies $\epsilon_n = 0$ and $\sum_{i=1}^{n} p'_i \cdot 2^{-e} = 1$.

Definition 7 *The* **projection** *of the 1-length signal ξ is the 1-length dyadic signal $p(\xi) = \sigma_1^{p'_1 \cdot 2^{-e}} \dots \sigma_n^{p'_n \cdot 2^{-e}}$ where e and p'_i are as in the lemma. The projection of a k-length signal ξ' is the dyadic signal $k \cdot p(k^{-1} \cdot \xi')$.*

To understand the meaning of this projection one may ask: "what does this projection lose in term of information about the signal ?" To answer, we notice that we temporary lose the length of the signal, as we project 1-length signals onto 1-signals. But we recover this information by multiplying again. This normalization operation preserves the different durations of the signal **up to their relative weight**. The real approximation lies in the foreclosing of the discontinuity points within dyadic intervals and the decision to take the upper letter on each of these intervals. We can speak about this as the first dyadic approximation **separating** the discontinuity points. Observe that relative weights are no longer exactly preserved, but they are **as far as** this first approximation permits it.

Representing signals by means of trees

The following classical definitions of trees are inspired from (Tho90).

Definition 8 *A $\mathcal{V}$-valued tree is an application $t : dom(t) \rightarrow \mathcal{V}$ where $dom(t) \subset \{0,1\}^*$ is a nonempty prefix-closed set which satisfies $wj \in dom(t) \wedge i < j \Rightarrow wi \in dom(t)$. Words in $dom(t)$ are called* **nodes** *in the tree and $t(w)$ is the* **label** *of node w. The set $fr(t) = \{w : wi \notin dom(t) \wedge w \in dom(t)\}$ of maximal elements of $dom(t)$ is the* **frontier** *of t, and its* **inner frontier** *is the set $fr^-(t) = \{w : \exists i, wi \in fr(t) \wedge w \in dom(t)\}$ Nodes of the frontier are called* **leaves** *of the tree.*

As an example of such a tree, one may give T on figure 2. Its domain is the set $\{\epsilon, 0, 00, 01, 000, 010, 1, 10\}$

Definition 9 *A* **binary tree** *is a tree where all nodes have either zero or two nodes. A* **quasi binary tree** *is a tree where nodes satisfy the binary tree property* **except** *at the inner frontier, where each node has exactly* **one** *child.*

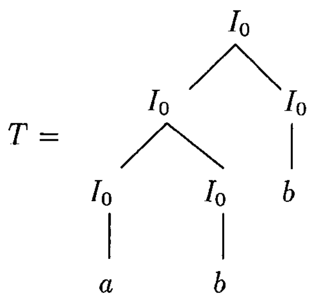

Figure 2: a quasi binary tree.

We now turn to the basic tree representation of signals, which is the syntactic part of the work. Instead of abruptly giving the pure syntactic part in a logic-like manner, let us try to highlight it with some operational considerations.

Suppose you want to represent the trace of a one-clock automaton run starting from q_0 with its clock c at τ_i and finishing at q_n with its clock between τ_j and τ_{j+1}. Basically, there are three levels of representation, from the simpler to the harder.

case 1: $i = j$ and no resetting state appears in the run. Then, one can use the dyadic projection of the trace to represent it as a quasi binary tree, with the information i at the root of the tree. The tree T of figure 2 is such an example of a representation; in particular it represents the signal of figure 1. As the projection identifies many signals, such a tree represents a set of signals. This remark is also valid in the two other cases.

case 2: $i < j$ and no resetting state appears in the run. Then the only possible evolution for c is to successively go through the values $\tau_{i+1} \ldots \tau_j$, and the trace ξ appears as the concatenation of trace runs $\xi_0, \ldots, \xi_{j-i}$ of the preceding form: each ξ_k is of length $\tau_{i+k+1} - \tau_{i+k}$ except the last one ($k = j-i$) the length of which is between 0 and $\tau_{j+1} - \tau_j$; moreover, each ξ_k is the trace of a partial run of the automaton starting at $c = \tau_{i+k}$ and finishing at $c = \tau_{i+k+1}$, excepting ξ_{j-i} the underlying run of which finishes with c between τ_j and τ_{j+1}.

case 3: $i \leq j$ and there are some resetting states in the run. We first mark the resetting state as $r_1, \ldots, r_m$ (note how we crucially need the state-output property to do that for an arbitrary trace), and we write ξ as the concatenation of $\xi_1, \ldots, \xi_m$ which are traces of runs containing resetting states at their limits only. For each ξ_k, the underlying run starts with the clock value $\tau_0 = 0$ and finishes with c between τ_l and τ_{l+1}, where τ_l depends on the length of ξ_k. These signals may be viewed as fitting the second case.

Definition 10 (signal representation) *Let $0 = \tau_0 < \tau_1 < \ldots < \tau_n < \tau_{n+1} = \infty$ an ordered list of constants and let $\mathcal{I} = \{{}_0I_i : 0 \leq i \leq n\} \cup \{\overline{{}_iI_j} : 0 \leq i < j \leq n\} \cup \{I_i : 0 \leq i \leq n\}$ a set of labels. An $\mathcal{I} \cup \Sigma$-valued quasi binary tree t with all its leaves labeled by a letter of Σ* **represents** *a set of signals in the following manner:*

case 1 : *t is $\{I_i\} \cup \Sigma$-valued. If $fr(t) = \{w_1, \ldots, w_s\}$ we put $i_1 = 1$ and $i_j = \min\{i : t(w_{i_{j-1}}) \neq t(w_{i_j})\}$ (so $\{i_j\}_j$ is the unique family s.t. $\sigma_1 = t(w_{i_1}) = t(w_1) = t(w_2) = \ldots = t(w_{i_2-1}) \neq \sigma_2 = t(w_{i_2}) = t(w_{i_2+1}) = \ldots = t(w_{i_3-1}) \neq \ldots \sigma_p = t(w_{i_p}) = \ldots t(w_s)$). Then t represents all the $(k - \tau_i)$-length dyadic signals where $\tau_i < k \leq \tau_{i+1}$ defined as:*

$$\xi = (k - \tau_i) \cdot \begin{pmatrix} {\sigma_1}^{2^{-|w1|+1} + \ldots + 2^{-|w_{i_2-1}|+1}} \\ {\sigma_2}^{2^{-|w_{i_2}|+1} + \ldots + 2^{-|w_{i_3-1}|+1}} \\ \cdots \\ {\sigma_p}^{2^{-|w_{i_p}|+1} + \ldots + 2^{-|w_s|+1}} \end{pmatrix}$$

case 2 : *t is obtained by substituting some $\{\overline{I_{i_k}}\} \cup \Sigma$-valued trees $t_1, \ldots, t_l$ at the leaves of a $\{\overline{{}_iI_j} : 0 \leq i < j \leq n\}$-valued binary tree. If the t_k's respectively represent sets of signals Ξ_k as in the first case then t represents all the concatenations of dyadic signals $\xi_1 \circ \xi_2 \circ \ldots \circ \xi_l$ where each ξ_k belongs to Ξ_k and the length of ξ_k is exactly $(\tau_{i_{k+1}} - \tau_{i_k})$ for $1 \leq k < l$ and arbitrary for $k = l$ (see figure 3). Notice how this case generalizes the precedent one by substituting a $\{I_i\} \cup \Sigma$-valued tree at the leaf of a single-noded tree.*

case 3: *t is obtained by substituting some trees $t_1, \ldots, t_m$ matching the precedent case definition at the leaves of a $\{{}_0I_i : 0 \leq i \leq n\}$-valued binary tree. If the t_k's respectively represent sets of signals Ξ_k as in the second case then t represents all the concatenations of signals $\xi_1 \circ \xi_2 \circ \ldots \circ \xi_m$ where each ξ_k belongs to Ξ_k.*

Remark:

1. $\mathcal{I} \cup \Sigma$-valued quasi binary trees not matching this case definition (for example the tree t s.t. $t(\epsilon) = \overline{{}_0I_3}$ and $t(0) = \overline{{}_0I_2}$) implicitly represent the void set of signals.

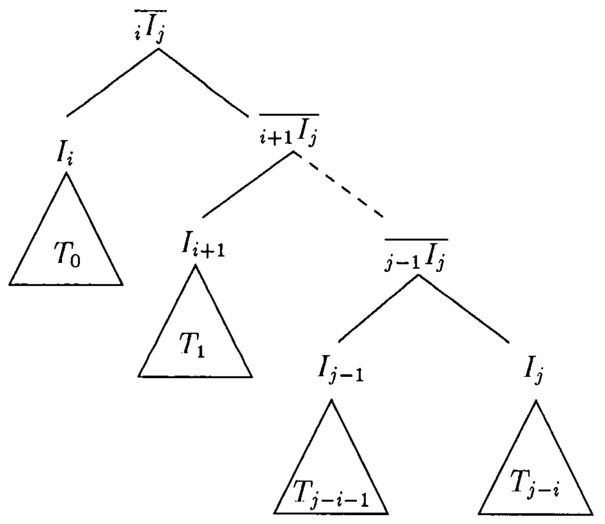

Figure 3: representation of the trace of a non resetting run.

2. The well-known property $\sum_{u\in fr(t)} 2^{-|u|} = 1$ for binary trees insures us that the length of dyadic signals $\sigma_1{}^{2^{-|w_1|+1}+\ldots+2^{-|w_{i_2-1}|+1}} \ldots \sigma_p{}^{2^{-|w_{i_p}|+1}+\ldots+2^{-|w_s|+1}}$ appearing in the first case of the definition is 1.

Regular tree languages

In this section, we define regular tree languages and we show how the trace of a simple partial run:

$$\begin{array}{ccccc} s & \longrightarrow & q_1 & \longrightarrow \; \ldots & f \\ \tau_i = t_0 & & \tau_i < t_1 & & t_n \le \tau_{i+1} \end{array}$$

which does not meet any resetting state can be represented by such a language.

Definition 11 *A (top-down)* **tree automaton on the signature** $\mathcal{V}$ *is a* 4*-tuple* $\hat{A} = (\hat{Q}, \hat{Q}_0, \hat{F}, \hat{\Delta})$ *where* $\hat{Q}$ *is a finite set of* states, $\hat{Q}_0, \hat{F} \subset \hat{Q}$ *are respectively the set of* initial *and* final *states. Letting* $\mathcal{B} = \hat{Q} \times \mathcal{V} \times \hat{Q} \times \hat{Q}$ *the set of* **binary rules** *and* $\mathcal{U} = \hat{Q} \times \mathcal{V} \times \hat{Q}$ *the set of* **unary rules**, $\mathcal{B} \cup \mathcal{U} \supset \hat{\Delta}$ *is the* **transition relation** *of the automaton. An* **execution** *of the tree automaton* A *on the binary tree* t *is a* $\hat{Q}$*-valued tree* r *such that* $dom^-(r) = dom(t)$ *with* $r(\epsilon) \in \hat{Q}_0$ *and* $(r(w), t(w), r(w0), r(w1)) \in \hat{\Delta}'$ *or* $(r(w), t(w), r(w0)) \in \hat{\Delta}'$ *for all* $w \in dom(t)$*; it is accepting if* $r(w) \in \hat{F}$ *for all* $w \in fr(r)$*. The set of tree* $T(A)$ **recognized** *by such an automaton contains the ones for which an accepting execution exists. Such a set is called* **regular**.

We are now ready to define our first regular tree languages.

Definition 12 *Let* $\mathcal{A} = (Q, \{c\}, \Delta, Q, Id, S, F)$ *be a one-clock timed automaton and let* $0 = \tau_0 < \tau_1 < \ldots < \tau_n < \tau_{n+1} = \infty$ *be the ordered list of constants appearing in its transition guards. For each* $0 \le i < n+1$*, we define:*

$$Q_i = \{q \in Q : q \xrightarrow{\phi} \textit{ and } \exists t \ge \tau_i \textit{ s.t. } t \models \phi\}$$

$$\textit{and: } Q^-_{i+1} = \{q \in Q : \xrightarrow{\phi} q \textit{ and } \exists t < \tau_{i+1} \textit{ s.t. } t \models \phi\} \cup S.$$

We define then, for all $0 \le i < n+1$ *and for all* $(s, f) \in Q_i \times Q^-_{i+1}$*, the tree automaton* $A_i^{s,f} = (\hat{Q}, \hat{Q}_0, \hat{\Delta}, \hat{F})$ *on the signature* $\{I_i\} \cup Q$ *whose state space is* $\hat{Q} = \{\langle q, q'\rangle : (q, q') \in Q \times Q\} \cup \{\overline{\langle q, q\rangle} : q \in Q\} \cup Q$ *and:*

initial state: $\hat{Q}_0 = \langle s, f\rangle$,

final states : $\hat{F} = \{OK\}$,

unary rules : *The set* $\mathcal{U}$ *is equal to*

$$\{(\langle q, q'\rangle, q, q) : q \xrightarrow{\phi} q' \in \Delta \wedge]\tau_i, \tau_{i+1}[\models \phi\} \\ \cup\{(\overline{\langle q, q\rangle}, q, q) : q \in Q\} \cup \{(q, q, OK)\},$$

binary rules : *The set* $\mathcal{B}$ *is equal to*

$$\{(\langle q, q'\rangle, I_i, \overline{\langle q, q\rangle}, \langle q, q'\rangle) : (q, q') \in Q^2\} \\ \cup\{(\langle q, q''\rangle, I_i, \langle q, q'\rangle, \langle q', q''\rangle) : (q, q', q'') \in Q^3\}.$$

Remark: A careful reading of the transition relation shows that this automaton can only accept quasi binary trees. Moreover, accepted trees have leaves labeled by letters of Q: the only rules leading to the final state are of the form $(., q, OK)$.

Now, how does such a tree automaton behave ? First, we note that $s \in Q_i$ means that the timed automaton may start a partial run from state s with a clock value greater than τ_i, and $f \in Q^-_{i+1}$ means that it may finish a partial run at state f with a clock value less than τ_{i+1}. Therefore, one can see the state $\langle s, f\rangle$ as a *question* from timed automaton to tree automaton like: " I am currently in state s. How can I reach state f in the remaining time?". Tree automaton tries to answer using the following recursive procedure:

- either wait for the beginning of the second half of the interval before doing anything; this is the first binary rule.
- or fire a transition to a state q_1 (closer to f than s) in the first half of the interval; this is second binary rule.

The tree automaton can delay its answer only on a half-interval and it cannot do it indefinitely (the accepting tree must be finite). Then, an accepting execution tree exhibits (at its inner frontier) a path from s to f and, for all intermediate states, the number of "wait" answers that the tree automaton gave, allowing different durations elapsed in each state.

We can now state the following long awaited lemma.

Lemma 3 (Key Lemma) *The trace of a partial run*

$$\begin{array}{ccccc} s & \longrightarrow & q_1 & \longrightarrow \; \ldots & f \\ t_0 = \tau_i & & t_1 = \tau_i + c_1 & & t_n < \tau_{i+1} \end{array}$$

which does not meet any resetting state is represented by some tree in $T(A_i^{s,f})$. Conversely, all the signals represented by a tree in $T(A_i^{s,f})$ are traces of such a partial run. We shall say that $A_i^{s,f}$ **describes the i^{th} local behavior** *of $\mathcal{A}$ between s and f.*

The global case

We now turn to the representation of non-resetting run traces with length greater than $\tau_{i+1} - \tau_i$: we slice such a signal into intervals $[\tau_i, \tau_{i+1}[$ and use the key lemma to locally represent the signal. Generating slicing trees can be done in the same spirit as in the local case : the global tree automaton may guess a path between $s \in Q_i$ and $f \in Q_{j+1}^-$. To find a path between q and f on an interval $[\tau_k, \tau_{j+1}[$, automaton slices this interval into $[\tau_k, \tau_{k+1}[$ and $[\tau_{k+1}, \tau_{j+1}[$, then it suggests to either **"do nothing"** on the first slice (i.e. wait in state q until the next slice) and recursively try to find a path from q to f on the second slice, or **trig** the right local grammar to find a path between q and q' in $[\tau_k, \tau_{k+1}[$ and recursively try to find a path from q' to f on the second slice. In technical terms, we **substitute** the appropriate local execution tree at the leaves of the global one, and closure under substitution is a well known regular grammars property.

The reset case is also similar in nature. We now have to design a "super-automaton" that can use resetting states to find a path between s with clock value 0 and f with clock value in $[\tau_j, \tau_{j+1}[$. This can be done by choosing a clock interval $[\tau_k, \tau_{k+1}[$ with $0 \leq k < j+1$, triggering the global grammar to find a non-resetting path between s with clock value 0 and r with clock value in $[\tau_k, \tau_{k+1}[$ and to guess recursively a path between r with clock value 0 and f with clock value in $[\tau_j, \tau_{j+1}[$. Note how this operation is intrinsically non-deterministic and how it can be iterated: supposing that the resetting state r is initial and reachable from itself on the time interval $[0, 1[$, one can construct a path from r to r by concatenating this little path an arbitrary number of times, allowing for always longer traces. This is in strong constrast with the global non-resetting case, where slicing trees always are in finite number. One will recognize here the effect of the regular **"star"operation**.

The previous discussion should convince the reader of the following theorem and corollary.

Theorem 1 *Given a one-clock automaton $\mathcal{A}$ there exists a regular tree language $T(A_j^{s,f})$ such that the traces of any run:*

$$\underset{t_0 = 0}{(s, v_0)} \xrightarrow{\delta_1} \underset{t_1}{(q_1, v_1)} \xrightarrow{\delta_2} \dots \xrightarrow{\delta_n} \underset{t_n}{(f, v_n)},$$

with $v_0 = 0$ and $\tau_j < v_n \leq \tau_{j+1}$ is represented by a tree in $T(A_j^{s,f})$. The converse also holds : any tree in this language represents such a signal.

Corollary 1 *The language of an arbitrary timed automaton is represented by a regular tree grammar.*

PROOF: Split the automaton by the procedure described in the third section of the paper and apply the theorem to each one-clock automaton. Taking the (finite) union of the regular tree languages $T(A_j^{s,f})$ for all initial states s, all final states f and all j between 0 and n, we obtain a regular tree language representing the language of each sub-automaton. We conclude then by closure of regular langages under intersection and renaming. □

Conclusion

In this paper, we have provided an interesting link between game theory approaches and hybrid system theory. Indeed, the construction of a succesful run for a timed automaton appears as a game played by the tree automaton, and our work shows what the basic strategies are that this automaton uses to play this game.

At a higher level, games between an automaton (playing transitions) and an opponent (playing directions in the constructed tree) are well-known, and strong results about determinacy (is there a winner?) and existence/synthesis of memory-bounded strategies are available (Tho95; GH82). We hope that these results could guide us to extend our considerations first to other classes of hybrid systems and second to infinite runs (by means of languages of infinite trees).

References

E. Asarin, P. Caspi, and O. Maler. A Kleene theorem for timed automata. In G. Winskel, editor, *LICS'97*, pages 161–171, 1997.

R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state operators. *Trans. of AMS*, 138:295–311, 1969.

Y. Gurevich and L.A. Harrington. Automata, trees and games. In *14th ACM Symposium on the theory of Computing*, pages 60–65, 1982.

J.E. Hopcroft and J.D. Ullman. *Introduction to Automata theory, Languages and Computation.* Addison-Wesley, 1979.

Y. N. Moschovakis. A game-theoretic modeling of concurrency. In *Fourth Annual Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1989.

A. Nerode, A. Yakhnis, and V. Yakhnis. Concurrent programs as strategies in games. In Springer-Verlag, editor, *Logic from Computer Science:Proceedings of a Workshop held November 13-17, 1989*, 1992.

W. Thomas. *Handbook of Theoretical Computer Science*, chapter Automata on Infinite Objects. J. van Leeuwen, 1990.

W. Thomas. On the synthesis of strategies in infinite games. In Springer-Verlag, editor, *STACS'95*, volume 900, pages 1-13. Lecture Notes in Computer Sciences, 1995.

C. Tomlin, J. Lygeros, and S. Sastry. Synthesizing controllers for nonlinear hybrid systems. In *Proceedings of Hybrid Systems: Computation and Control*, Berkeley, April 1998.