Electronic Notes in Theoretical Computer Science 68 No. 5 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume68.html 18 pages

Balanced timed regular expressions [1]

Eugene Asarin [2] Cătălin Dima [3]

VERIMAG, 2 av. de Vignate, 38610 Gières, France

Abstract

Several classes of regular expressions for timed languages accepted by timed automata have been suggested in the literature. In this article we introduce *balanced timed regular expressions with colored parentheses* which are equivalent to timed automata, and, differently from existing definitions, do not refer to clock values, and do not use additional operations such as intersection and renaming.

1 Introduction

Regular expressions are an important and convenient formalism to specify sets of discrete behaviors. The lack of such a language-based algebraic formalism for timed behaviors motivated several researchers to look for some variants of "timed regular expressions" equivalent to timed automata. The formalisms suggested in the literature give some solutions to this problem, but none of them is as perfect as classical "discrete" regular expressions of Kleene. In fact some of them [5,6,7] make use of information external to the language (such as clock values), while others [3,4] use heavy operations such as intersection and renaming.

In this paper we suggest a new approach to this problem. We introduce *balanced timed regular expressions with colored parentheses* which are equivalent to timed automata, do not refer to clock values, and do not use "bad" operations. The price to pay is a two-stage semantics of our expressions and a non-trivial algorithm for checking whether an expression is syntactically correct.

The structure of the paper is the following: in section 2 we recall some definitions concerning timed automata and timed languages and state the key technical result: Theorem 2.1 on transforming timed automata to a special form. In Section 3 we recall timed regular expressions from [4], discuss their

[1] Partially supported by the European community project IST-2001-35304 Ametist

[2] Email: asarin@imag.fr. Home page: www-verimag.imag.fr/~asarin.
[3] Email: dima@enseirb.fr. Home page: www.enseirb.fr/~dima.

drawbacks, and come up with our definition of *balanced timed regular expressions* and their semantics. At the end of this section we state the main result of the paper: balanced timed regular expressions are as expressive as timed automata. In section 4 we give an algorithm for checking correctness of an expression.

We are thankful to Christian Boitet for a motivating question and to Paul Caspi, Oded Maler and Matthieu Moy for useful discussions.

2 Timed automata

Behaviors of timed systems can be modeled by **timed words** over a set of symbols Σ. A timed word is a finite sequence of nonnegative numbers and symbols from Σ. For example, the sequence $1.2\,a\,1.3\,b$ denotes a behavior in which an action a occurs 1.2 time units after the beginning of the observation, and after another 1.3 time units action b occurs. The set of timed words over Σ can be organized as a monoid, and be represented as the *direct sum* of the monoid of nonnegative numbers $(\mathbb{R}_{\geq 0}, +, 0)$ and the free monoid $(\Sigma^*, \cdot, \varepsilon)$ [4,11]. We denote this monoid as $\mathsf{TW}(\Sigma)$. Note that in this monoid, concatenation of two reals amounts to summation of the reals. Thence, $a\,1.3 \cdot 1.7\,b = a(1.3 + 1.7)b = a\,3\,b$. The length $\ell(w)$ of a timed word w is the sum all the reals in it, e.g. $\ell(1.2\,a\,1.3\,b) = 1.2 + 1.3 = 2.5$.

A **timed automaton** [1] is a tuple $\mathcal{A} = (Q, \mathcal{X}, \Sigma, \delta, Q_0, Q_f)$ where Q is a finite set of *states*, $\mathcal{X}$ is a finite set of *clocks*, Σ is a finite set of *action symbols*, $Q_0, Q_f \subseteq Q$ are sets of *initial*, resp. *final* states, and δ is a finite set of tuples (*transitions*) (q, C, a, X, r) where $q, r \in Q$, $X \subseteq \mathcal{X}$, $a \in \Sigma \cup \{\varepsilon\}$ and C is a finite conjunction of *clock constraints* of the form $x \in I$, where $x \in \mathcal{X}$ and $I \subseteq [0, \infty[$ is an interval with integer (or infinite) bounds.

For each transition $(q, C, a, X, r) \in \delta$, the component C is called the *guard* of the transition, a is called the *action label* of the transition, and X is called the *reset component* of the transition. We will usually order the set of clocks $\mathcal{X} = \{x_1, \ldots, x_n\}$, and then identify each reset component X with the subset of indices of the clocks in X, that is, with $\big\{i \mid i \in [n], x_i \in X\big\}$ (here $[n]$ stands for $\{1, \ldots, n\}$).

The semantics of a timed automaton is given in terms of a *timed transition system* $\mathcal{T}(\mathcal{A}) = (\mathcal{Q}, \theta, \mathcal{Q}_0, \mathcal{Q}_f)$ where $\mathcal{Q} = Q \times \mathbb{R}^n_{\geq 0}$, $\mathcal{Q}_0 = Q_0 \times \{\mathbf{0}_n\}$, $\mathcal{Q}_f = Q_f \times \mathbb{R}^n_{\geq 0}$ and

$$
\begin{aligned}
\theta = \big\{(q, v) \xrightarrow{t} (q, v') \mid v'_i = v_i + t,\ \forall i \in [n]\big\} \cup \\
\big\{(q, v) \xrightarrow{a} (q', v') \mid \exists (q, C, a, X, q') \in \delta \text{ such that } v \models C \text{ and for all } i \in [n], \\
\text{if } i \in X \text{ then } v'_i = 0 \text{ and if } i \notin X \text{ then } v'_i = v_i\big\}
\end{aligned}
$$

Informally, the automaton can make t-transitions representing time-passage in a state, in which all clocks advance by t, and discrete transitions, in which state changes. The discrete transitions are enabled when the "current clock valuation" v satisfies the guard C of a certain tuple $(q, C, a, X, r) \in \delta$, and when they are executed, the clocks in the "reset component" X are set to zero.

A **run** in $\mathcal{T}(\mathcal{A})$ is a chain $(q^0, v^0) \xrightarrow{\xi_1} (q^1, v^1) \xrightarrow{\xi_2} \ldots \xrightarrow{\xi_k} (q^k, v^k)$ of transitions from θ. An **accepting run** in $\mathcal{T}(\mathcal{A})$ is a run which starts in $\mathcal{Q}_0$ and ends in $\mathcal{Q}_f$ (the last transition should not be a t-transition). The **accepted language** of $\mathcal{A}$ is then the set of timed words which label some accepting run of $\mathcal{T}(\mathcal{A})$. Two timed automata are called **equivalent** iff they have the same language.

The first theorem gives a "normal form" to which each timed automaton can be brought. For it, we need the following notations and conventions: for each clock $x_i \in \mathcal{X}$ and each transition $\tau = (q, C, a, X, r) \in \delta$, if $i \in X$ then we say that x_i *is reset on* τ. If the guard C contains a constraint $x_i \in I$ for some interval I, then we say that τ *checks* x_i, and we also write $(x_i \in I) \in C$. Note that the *true* guard contains no constraint.

Theorem 2.1 *Any timed automaton $\mathcal{A}$ is equivalent to a timed automaton $\overline{\mathcal{A}}$ in which on each accepting run, each clock is checked exactly once after each reset.*

Proof. We will decompose the construction into two steps as follows:

(i) We transform the given automaton into an automaton in which, each clock is checked against *the same* interval I, wherever it is checked.

(ii) We obtain the desired construction by splitting each clock x into two copies such that each copy is checked only once after each reset.

The first construction is accomplished along the following ideas: for each clock $x \in \mathcal{X}$ and interval I for which $(x \in I)$ occurs on some guard, we create a new clock x_I. The set of all clocks x_I will replace the clock x, in the following sense: each time x is reset, we reset all clocks x_I; then, each time $(x \in I)$ occurs on the guard of some transition, we replace this constraint with $(x_I \in I)$.

More formally we replace each transition $(q, C, a, X, r) \in \delta$ by the transition (q, C', a, X', r) with

$$C' = \bigwedge_{(x \in I) \in C} (x_I \in I)$$

and with $X' = \{x_I \mid x \in X\}$.

If we want to be practical, we should also remove the "unused clocks" as described in [10] (see also [9]).

The second construction is an adaptation of the convexity-based techniques of [4,2]. The rough idea is that in a chain of transitions which all contain $(x \in I)$ and do not reset x, all but the first and the last are "redundant". That happens because if the value of clock x on the first and on the last transition of the chain is in the interval I, then it must be in the interval

I throughout all the behavior of the timed automaton in between these two transitions – as a consequence of the convexity of I.

Therefore, we create two copies of each clock $x \in \mathcal{X}'$, denote them x^1 and x^2, and utilize them as follows: we reset both on each transition on which x is reset, we check $(x^1 \in I)$ on the first transition after the reset on which $(x \in I)$ occurs, we ignore all the other checks for $(x \in I)$ before the next reset, with the exception of the *last*, on which we check $(x^2 \in I)$. This last transition before a reset of x on which the constraint $(x \in I)$ occurs is not deterministically found, but rather "guessed".

The states of the resulting automaton will be tuples (q, ϕ) in which $q \in Q'$ and $\phi : \mathcal{X}' \to \{0, 1, 2\}$ gives our guess for the utilization of each clock x. An attribute $\phi(x) = 0$ means that, since the last reset for x we have never encountered the constraint $(x \in I)$ (remind that each clock is tested against a *unique* interval!). The attribute is set to $\phi(x) = 1$ the first time when such a constraint is met. It then remains 1 until we "guess" that, from now on and before the next reset for x, we will never take a transition with $(x \in I)$. Of course, such a guess is made on a transition on which $(x \in I)$ occurs, and this guess sets x's attribute to $\phi(x) = 2$.

Formally, for each transition $(q, C, a, X, r) \in \delta'$ and for each state (q, ϕ) in the resulting automaton we draw a transition $\big((q, \phi), \overline{C}, a, Y, (r, \psi)\big)$ if and only if the following conditions are satisfied:

- $x \in X$ iff $x^1, x^2 \in Y$.
- If $\phi(x) = 2$ then $(x \in I) \notin C$ and also $(x^1 \in I), (x^2 \in I) \notin \overline{C}$.
- If $x \in X$ then $\phi(x) = 2$ and $\psi(x) = 0$.
- If $(x \in I) \in C$ then $\phi(x) \neq 2, \psi(x) \neq 0$.
- If $(x \in I) \in C$ and $\phi(x) = 0$ then either $\psi(x) = 1$ and $(x^1 \in I) \in \overline{C}$, or $\psi(x) = 2$ and $(x^1 \in I), (x^2 \in I) \in \overline{C}$.
- If $(x \in I) \in C$ and $\phi(x) = 1$ then either $\psi(x) = 1$ and $(x^1 \in I), (x^2 \in I) \notin \overline{C}$, or $\psi(x) = 2$ and $(x^2 \in I) \in \overline{C}$.

□

3 Expressions

3.1 Expressions from [4]

The class of *timed regular expressions* is built using the following grammar:

$$E ::= 0 \mid \varepsilon \mid \underline{\mathbf{t}}z \mid E + E \mid E \cdot E \mid E^* \mid \langle E \rangle_I, \tag{1}$$

where $z \in \Sigma \cup \{\varepsilon\}$ and I is an interval.

The semantics of timed regular expressions is in terms of timed words:

$$\|\underline{\mathbf{t}}z\| = \{tz \mid t \in \mathbb{R}_{\geq 0}\} \qquad \|E_1 + E_2\| = \|E_1\| \cup \|E_2\|$$
$$\|E_1 \cdot E_2\| = \|E_1\| \cdot \|E_2\| \qquad \|\langle E\rangle\|_I = \{\sigma \in \|E\| \mid \ell(\sigma) \in I\}$$
$$\|E^*\| = \|E\|^* \qquad \|0\| = \emptyset \ , \qquad \|\varepsilon\| = \{\varepsilon\}$$

We abuse notation and write $\langle E\rangle_\alpha$ for $\langle E\rangle_{[\alpha,\alpha]}$. We also denote $\Sigma_{\underline{\mathbf{t}}} = \Sigma \cup \{\underline{\mathbf{t}}\}$.

The following theorem shows a nice relationship between timed automata and timed regular expressions:

Theorem 3.1 ([4]) *The class of timed languages accepted by timed automata equals the class of timed languages accepted by timed regular expressions with* intersection *and* renaming*, that is, expressions generated by the grammar*

$$E ::= 0 \mid \varepsilon \mid \underline{\mathbf{t}}z \mid E + E \mid E \cdot E \mid E^* \mid \langle E\rangle_I \mid E \wedge E \mid [a \mapsto z]E,$$

where $z \in \Sigma \cup \{\varepsilon\}, a \in \Sigma$.

We interpret $\wedge$ as intersection, and $[a \mapsto z]$ as renaming of any occurrence of the symbol a with the symbol z. For example, $\big\|[a \mapsto b](\underline{\mathbf{t}}a)\| = \|\underline{\mathbf{t}}b\|$.

It was shown in [3] that intersection is necessary for representing timed automata. The example there is the timed regular expression $\underline{\mathbf{t}}a\langle\underline{\mathbf{t}}b\underline{\mathbf{t}}c\rangle_1 \wedge \langle\underline{\mathbf{t}}a\underline{\mathbf{t}}b\rangle_1\underline{\mathbf{t}}c$, which cannot be expressed without conjunction. The timed language accepted by this expression is

$$L_0 = \{t_1 a t_2 b t_3 c \mid t_1 + t_2 = 1, t_2 + t_3 = 1\}$$

Moreover, in [13] it was shown that renaming is necessary too, his example being the timed automaton in the figure 1.

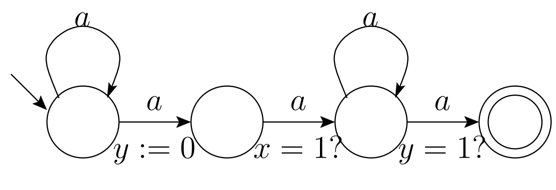

Fig. 1.

The language of this automaton equals

$$[x \mapsto a]\big((\underline{\mathbf{t}}a)^*\langle\underline{\mathbf{t}}x(\underline{\mathbf{t}}a)^*\rangle_1 \wedge \langle(\underline{\mathbf{t}}a)^*\underline{\mathbf{t}}x\rangle_1(\underline{\mathbf{t}}a)^*\big).$$

These results show that, in spite of their ease in use, timed regular expressions suffer from some expressiveness problems.

3.2 Colored parentheses

In [4], the authors suggest that, by employing "colored" parentheses, these drawbacks might be overcome. For example, L_0 would be specified by the

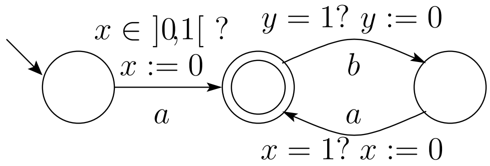

Fig. 2.

expression $\langle \underline{\mathbf{t}}a \lfloor \underline{\mathbf{t}}b \rangle_1 \underline{\mathbf{t}}c \rfloor_1$. Similarly, the language of the timed automaton in Figure 1 would be specified by $\langle (\underline{\mathbf{t}}a)^* \lfloor \underline{\mathbf{t}}a \rangle_1 (\underline{\mathbf{t}}a)^* \rfloor_1$. Note that these expressions no longer use intersection or renaming.

But then, if we want to specify also cyclic behaviors, we cannot cope with the simple automaton on the figure 2. The expression with intersection for its language is

$$\langle \underline{\mathbf{t}}a \rangle_{]0,1[} \langle \underline{\mathbf{t}}b\underline{\mathbf{t}}a \rangle_1^* \wedge \langle \underline{\mathbf{t}}a\underline{\mathbf{t}}b \rangle_1^* \underline{\mathbf{t}}a$$

We conjecture that the semantics of this formula cannot be expressed with colored parentheses (in the sense of the suggestion from [4]).

Another problem with colored parentheses is that the set of syntactically correct expressions is non-context-free. Indeed, let us consider $\mathcal{J}$ an index set (the set of *colors*) and $card(\mathcal{J})$ sets of matching parentheses:

$$P_i = \{ `\langle^i`, `\rangle_I^i` \mid I \text{ interval }, I \subseteq [0, \infty[\} \text{ for all } i \in \mathcal{J} \tag{2}$$

$$\Pi = \bigcup_{i \in \mathcal{J}} P_i. \tag{3}$$

We may define $card(\mathcal{J})$ *deletion morphisms* (or *color filters*), $(\eta_i)_{i \in \mathcal{J}}$, $\eta_i : \Pi \to P_i$, each η_i deleting all parentheses not in P_i. E.g., for $P_{blue} = \{ `\langle`, `\rangle_I` \mid I \subseteq [0, \infty[\}$ and $P_{red} = \{ `\lfloor`, `\rfloor_I` \mid I \subseteq [0, \infty[\}$, $\eta_{blue}\big(\langle \lfloor \rangle_1 \langle \rfloor_1 \rangle_1 \big) = \langle \rangle_1 \langle \rangle_1$.

For each set of parentheses $P_i = \{ `\langle^i`, `\rangle_I^i` \mid I \text{ interval}, I \subseteq [0, \infty[\ \}$, we denote by Δ_i the set of words with balanced parentheses [4] over P_i, which is generated by the following context-free grammar:

$$S ::= \varepsilon \mid \langle^i S \rangle_I^i \mid SS$$

The language of *balanced parentheses* over $\bigcup_{i=1}^n P_i$ is defined as follows:

$$L_{\mathrm{par}} = \Big\{ w \in \Pi^* \mid \text{ for each } i \in [n], \eta_i(w) \in \Delta_i \Big\}$$

This language is unfortunately context-sensitive for $card(\mathcal{J}) \geq 2$: just consider the intersection of L_{par} with $\big(\langle\big)^* \big(\lfloor\big)^* \big(\rangle_1\big)^* \big(\rfloor_1\big)^*$, which gives a language of the form $\{a^k b^l c^k d^l \mid k, l \in \mathbb{N}\}$, which is an easy prey to the Bar-Hillel (pumping) lemma for context-free languages [14]. In a related paper [12], one of the authors investigates on the possibility to define regular expressions with colored parentheses by using a different concatenation operation.

[4] This is a slight generalization of the notion of Dyck languages [14].

3.3 Balanced timed expressions

We explain our approach with an example: consider again the language of the automaton from the figure 2. This language can be regarded as the union of all the "word expressions" of the kind

$$\langle \lfloor \underline{\mathbf{t}} a \rfloor_{]0,1[} \lfloor \underline{\mathbf{t}} b \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1 \lfloor \underline{\mathbf{t}} b \rangle_1 \ldots$$

The following expression generates all these "word expressions":

$$E = \lfloor \underline{\mathbf{t}} a \rfloor_{]0,1[} + \langle \lfloor \underline{\mathbf{t}} a \rfloor_{]0,1[} \big(\lfloor \underline{\mathbf{t}} b \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1 \big)^* \lfloor \underline{\mathbf{t}} b \rangle_1 \underline{\mathbf{t}} a \rfloor_1 \tag{4}$$

Note that the resulting expression E contains the subexpression $\lfloor \underline{\mathbf{t}} b \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1$ in which the parentheses *are not balanced* – the first "blue" parenthesis is the right one. In order to give a sense to E and to other expressions of this kind we adopt a *two-step semantics* approach, in which we first build the classical semantics of the expression – i.e. a set of words over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$. In the second step we give timed semantics to these words, provided they have well-balanced parentheses. Hence, a balanced timed regular expression will be defined as an expression over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ which generates only words with well-balanced parentheses, and its timed semantics will be the union of the semantics of these words.

Observe that this process is not "compositional", that is, we do not define the semantics of a balanced timed regular expression by induction on their structure. In fact, even the definition of the class of regular expressions with balanced parentheses is not a "structural" once. This is the point of difference with [12].

The problem of checking whether an expression generates only well-balanced words is the subject of the last section.

Formalization

For the sequel, we will work with n sets of colored parentheses P_i, as defined in (2) on the previous page. That is, we assume $\mathcal{J} = \{1, \ldots, n\}$. We will utilize here the deletion morphism $\eta_i : (\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^* \to P_i^*$, which deletes all symbols not in P_i, and the "partial" deletion morphism $\overline{\eta}_i : (\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^* \to P_i^*$, which deletes from each word the symbols not in P_i or not in $\Sigma_{\underline{\mathbf{t}}}$. For example,

$$\begin{aligned} \eta_{blue}\big(\lfloor \underline{\mathbf{t}} a \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1 \big) &= \rangle_1 \langle \\ \overline{\eta}_{blue}\big(\lfloor \underline{\mathbf{t}} a \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1 \big) &= \underline{\mathbf{t}} a \rangle_1 \langle \underline{\mathbf{t}} a \end{aligned}$$

Definition 3.2 A **balanced word** over $\Sigma_{\underline{\mathbf{t}}}$ with parentheses from Π is a word $w \in (\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^*$ such that $\eta_i(w) \in \Delta_i$ for all i. The set of balanced words over $(\Sigma_{\underline{\mathbf{t}}} \cup \Pi)^*$ is denoted $\mathcal{W}_\Pi(\Sigma)$.

Note that, for each $1 \leq i \leq n$ and $w \in \mathcal{W}_\Pi(\Sigma)$, $\overline{\eta}_i(w)$ is a *timed regular expression.*

We would like to define the **timed semantics** of a balanced word by intersecting the semantics w.r.t. each color, in the following sense:

$$[|w|] = \bigcap_{i\in\mathcal{J}} \|\overline{\eta}_i(w)\|$$

Unfortunately this works only when each parenthesis is "next to" a symbol in Σ. For an expression like $\lfloor\underline{\mathbf{t}}\rfloor_1\langle\underline{\mathbf{t}}\rangle_1$ this definition would yield the interval $\big[1,\infty\big[$ for semantics, whereas what we want is the set $\{2\}$.

Therefore we need to "pin" each parenthesis to a "virtual" action $x \notin \Sigma_{\underline{\mathbf{t}}}$, take the above definition and then erase the x symbol from the resulting timed words.

To this end, we need several word morphisms:

(i) The morphism $\phi_x : \Sigma_{\underline{\mathbf{t}}} \cup \Pi \to \Sigma_{\underline{\mathbf{t}}} \cup \Pi \cup \{x\}$ defined by $\phi(a) = a$ for all $a \neq \underline{\mathbf{t}}$ and $\phi(\underline{\mathbf{t}})_{=}x\underline{\mathbf{t}}$.

(ii) For each $i \in \mathcal{J}$, the extension of the deletion morphism $\overline{\eta}_i$ to $\Sigma_{\underline{\mathbf{t}}}\cup\Pi\cup\{x\}$ which, as before, deletes all parentheses not in P_i. This extension is denoted $\overline{\eta}_i$ too.

The corresponding definition is then the following:

Definition 3.3 The **semantics** of a balanced word $w \in \mathcal{W}_\Pi(\Sigma)$ is the set

$$\|w\| = \big[x \mapsto \varepsilon\big]\Big(\bigcap_{i\in\mathcal{J}}[|\phi(\overline{\eta}(w))|]$$

Our *regular expressions with colored parentheses* are classical regular expressions over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$, that is, generated by the grammar

$$E ::= 0 \mid a \mid \underline{\mathbf{t}} \mid \langle^i \mid \rangle^i_I \mid E + E \mid E \cdot E \mid E^*$$

where $a \in \Sigma$ and $\langle^i, \rangle^i_I \in P_i$, for some $1 \leq i \leq n$. As classical regular expressions, they have a *word semantics* in terms of languages over $(\Sigma_{\underline{\mathbf{t}}}\cup\Pi)^*$:

$$\begin{aligned} |a| &= \{a\} & |\underline{\mathbf{t}}| &= \{\underline{\mathbf{t}}\} \\ |0| &= \emptyset & |\langle^i| &= \{\langle^i\}, \\ |\rangle^i_I| &= \{\rangle^i_I\} & |E_1 + E_2| &= |E_1| \cup |E_2| \\ |E_1 \cdot E_2| &= |E_1| \cdot |E_2| & |E^*| &= |E|^* \end{aligned}$$

Definition 3.4 A **balanced (timed) regular expression** with colored parentheses in Π is a regular expression with colored parentheses whose word semantics contains only balanced words.

The *timed* semantics of a balanced regular expression is then the union of the timed semantics of each balanced word in its (word) semantics:

$$\|E\| = \bigcup\big\{\|w\| \mid w \in |E|\big\}$$

This definition already points out the difficulty of constructing regular expressions over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ that correspond to timed automata: we first need to build classical semantics of regular expressions in order to check whether all the words in this semantics have balanced parentheses. Only after this validation we may construct the timed semantics of the given expression.

Thanks to the particular form of timed automata provided by Theorem 2.1, we may prove the following form of the Kleene theorem:

Theorem 3.5 (Kleene theorem for timed automata) *The class of timed languages accepted by timed automata equals the class of timed languages which are the timed semantics of some balanced regular expression.*

Proof. We transform each timed automaton in the special form provided by Theorem 2.1 into a finite automaton whose transitions are labeled with *words* over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ (such automata are called *extended*, e.g., in [16]). The rough idea is very simple: each reset for clock x_i is transformed into the parenthesis $\langle^i$, while each clock check $x_i \in I$ is transformed into the parenthesis $\rangle^i_I$. We apply next the classical Kleene theorem to convert this automaton into a regular expression over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$. This expression is balanced since the automaton is in the special form of Theorem 2.1.

The proof of the reverse implication is delayed for the last section as it relies on results proved below in section 4.

$\square$

4 Checking regular expressions for balance

In this section we give an algorithm for deciding whether an expression E over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ is balanced. The idea is to associate special attributes to each sub-expression of E. These attributes represent the number of left and right parentheses of each color which are not balanced. The algorithm computes recursively these attributes for all the sub-expressions and decides that E is balanced if and only if all its attributes are zero.

In this section we use the "nonnegative" subtraction ("monus"),

$$a \dot{-} b = \begin{cases} a - b & \text{iff } a - b \geq 0 \\ 0 & \text{iff } a - b < 0 \end{cases}$$

We extend this operation and the addition to sets of naturals, in the straightforward manner: given two sets of natural numbers $A, B \subseteq \mathbb{N}$, we put $A + B = \{a + b \mid a \in A, b \in B\}$ and $A \dot{-} B = \{a \dot{-} b \mid a \in A, b \in B\}$.

Proposition 4.1 *Consider the binary operation $\ominus$ on $\mathbb{N} \times \mathbb{N}$, defined by*

$$(a, b) \ominus (c, d) = \big((a \dot{-} d) + c, (d \dot{-} a) + b\big)$$

(The parentheses are needed as $\dot{-}$ and $+$ are not "associative" with each other, like $-$ and $+$ are). Then $\big(\mathbb{N}\times\mathbb{N},\ominus,(0,0)\big)$ is a monoid.

Furthermore, the mapping $\varphi:\mathbb{N}\times\mathbb{N}\to\mathbb{Z}$ defined by $\varphi(a,b)=a-b$, is a monoid morphism to the target monoid $(\mathbb{Z},+,0)$.

We extend all these operations onto sets of *n-tuples* of naturals (resp. integers). As a corollary of Proposition 4.1, $\big(\mathbb{N}^n\times\mathbb{N}^n,\ominus,(\mathbf{0}_n,\mathbf{0}_n)\big)$ is also a monoid and the mapping $\varphi_n:\mathbb{N}^n\times\mathbb{N}^n\to\mathbb{Z}^n$ defined by $\varphi_n(\overline{a},\overline{b})=\overline{a}-\overline{b}$ is a monoid morphism.

We also use the "unit" vectors $e_i=(e_i^1,\ldots,e_i^n)\in\mathbb{N}^n$, given by

$$e_i^j=\begin{cases}1 & \text{for } j=i\\ 0 & \text{for } j\neq i\end{cases}$$

And finally, for any set $A\subseteq\mathbb{N}^n\times\mathbb{N}^n$ let

$$A^{k\ominus}=\underbrace{A\ominus\ldots\ominus A}_{k\text{ times}}$$

We proceed now to our construction of attributes: we first associate, to each word $w\in(\Sigma_{\underline{\mathbf{t}}}\cup\Pi)^*$, two naturals $l(w)$ and $r(w)$ which, intuitively, give the number of left, resp. right parentheses which are not balanced in w. The formal definition proceeds by induction on the length of the word:

$$\begin{aligned} l(\varepsilon)=r(\varepsilon)&=\mathbf{0}_n, & l(w\xi)=l(w),\ r(\xi w)&=r(w),\ \forall\xi\in\Sigma_{\underline{\mathbf{t}}}\\ l(w\langle^i)&=l(w)+e_i, & r(\langle^i w)&=r(w)\dot{-}e_i\\ l(w\rangle^i_I)&=l(w)\dot{-}e_i, & r(\rangle^i_I w)&=r(w)+e_i \end{aligned}$$

Together, l and r define a mapping from $(\Sigma_{\underline{\mathbf{t}}}\cup\Pi)^*$ to $\mathbb{N}\times\mathbb{N}$, denoted in the sequel by lr:

$$\mathsf{lr}(w)=\big(l(w),r(w)\big)$$

Proposition 4.2 $\mathsf{lr}:(\Sigma_{\underline{\mathbf{t}}}\cup\Pi)^*\to\mathbb{N}\times\mathbb{N}$ *is a monoid morphism, i.e.* $\mathsf{lr}(w_1w_2)=\mathsf{lr}(w_1)\ominus\mathsf{lr}(w_2)$.

Remark 4.3 w is a balanced word iff $l(w)=r(w)=0$. Or, in other words, the set of balanced words is the *kernel* of the morphism lr, that is, $\mathsf{lr}^{-1}(0,0)$.

The next step is to extend lr to *regular expressions*, by the rule

$$\mathsf{lr}_e(E)=\bigcup\big\{\mathsf{lr}(w)\mid w\in|E|\big\}$$

The following result represents the basis of our technique:

Proposition 4.4 *E is a balanced timed regular expression if and only if*

$$\mathsf{lr}_e(E)=\big\{(\mathbf{0}_n,\mathbf{0}_n)\big\}.$$

Our first aim is to show that lr_e can be computed *by structural induction* on the expression E. The following property paves the way for this approach:

Proposition 4.5 *For any regular expressions E, E_1, E_2 over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$,*

$$\begin{aligned} \mathsf{lr}_e(E_1 + E_2) &= \mathsf{lr}_e(E_1) \cup \mathsf{lr}_e(E_2) \\ \mathsf{lr}_e(E_1 \cdot E_2) &= \mathsf{lr}_e(E_1) \ominus \mathsf{lr}_e(E_2) \\ \mathsf{lr}_e(E^*) &= \bigcup_{k \in \mathbb{N}} \mathsf{lr}_e(E)^{k\ominus} \end{aligned}$$

For example, for $P_1 = \big\{ `\langle`, `\rangle_I` \mid I \subseteq [0,\infty[\big\}$ and $P_2 = \big\{ `\lfloor`, `\rfloor_I` \mid I \subseteq [0,\infty[\big\}$,

$$\mathsf{lr}_e\Big(\big(\, \underline{\mathbf{t}} a \lfloor b \, \big)^* \Big) = \left\{ \left(\begin{pmatrix} 0 \\ k \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right) \middle| \, k \in \mathbb{N} \right\} \tag{5}$$

$$\begin{aligned} &\mathsf{lr}_e\Big(\big(\, \lfloor \underline{\mathbf{t}} a \rangle_1 \langle \underline{\mathbf{t}} a \rfloor_1 + \rfloor_2 \underline{\mathbf{t}} a \lfloor \, \big)^* \Big) = \\ &\qquad = \left\{ \left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right), \left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right), \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) \right\} \end{aligned} \tag{6}$$

The results from Propositions 4.4 and 4.5 is not sufficient for deciding whether an expression E is balanced. The reason is that star might generate *infinite* sets of tuples of naturals – as we can see in example (5) above. We need therefore a different technique to check expressions for balance.

Let us make first the following observation: whenever $A \subseteq \mathbb{N} \times \mathbb{N}$ is an infinite set, for any set $B \subseteq \mathbb{N} \times \mathbb{N}$, $A \ominus B$ will be an infinite set. This means that, in our inductive calculus on the structure of a regular expression, whenever we obtain an infinite $\mathsf{lr}_e(E')$ for some subexpression E' of E, we should halt the computation and decide that the whole expression is non-balanced. Therefore, it only remains to find out when does $\mathsf{lr}_e(E^*)$ consist of an infinite set. But we may halt even earlier, as we will show intuitively in the sequel.

Note, from example (6) above, that, when in an expression E the difference between the left and the right parentheses of a certain kind is not zero, then $\mathsf{lr}_e(E^*)$ will be an infinite set. On the contrary, whenever $\mathsf{lr}_e(E)$ contains only tuples which give the same difference between left and right parentheses, $\mathsf{lr}_e(E^*)$ is finite.

Consider then, for each expression E, the set

$$\varphi_e(E) = \big\{ l(w) - r(w) \mid w \in |E| \big\} = \big\{ \overline{a} - \overline{b} \mid (a,b) \in \mathsf{lr}_e(E) \big\}$$

which holds the set of differences between lest and right parentheses in the words generated by E.

Observe then that, if, for some expression E_1, $card(\varphi_e(E_1)) \geq 2$, then for any expression E_2 which contains E_1 as a subexpression, $card(\mathsf{lr}_e(E_2)) \geq 2$

and therefore E_2 cannot be balanced. That is, we may halt our algorithm as soon as we obtain $card(\varphi_e(E')) \geq 2$ for some subexpression E'.

The rest of this section gives the formalization of these observations.

Recall that a *Kleene algebra* [8] is an algebra $\big(A, \cup, \cdot, *, \emptyset, \{e\}\big)$ that verifies all the equations valid in the structure $\big(\mathcal{P}(\Sigma^*), \cup, \cdot, *, \emptyset, \{\varepsilon\}\big)$. Furthermore, remind that each monoid $(M, e, \cdot)$ naturally generates a Kleene algebra $\big(\mathcal{P}(M), \cup, \cdot, *, \emptyset, \{\varepsilon\}\big)$, by putting, for each $S \subseteq M$,

$$S^* = \bigcup\nolimits_{n\in\mathbb{N}} \Big(\underbrace{S \cdot \ldots \cdot S}_{n \text{ times}} \Big)$$

Therefore, the monoid $\big(\mathbb{N}^n \times \mathbb{N}^n, \ominus, (\mathbf{0}_n, \mathbf{0}_n)\big)$ generates a Kleene algebra, in which the star is denoted $\circledast$:

$$A^{\circledast} = \{(\mathbf{0}_n, \mathbf{0}_n)\} \cup \bigcup_{k\in\mathbb{N}} A^{k\ominus}$$

Furthermore, the monoid $(\mathbb{Z}^n, +, \mathbf{0}_n)$ generates, on its turn, a Kleene algebra, in which

$$A^* = \{\mathbf{0}_n\} \cup \bigcup_{k\in\mathbb{N}} \Big(\underbrace{A + \ldots + A}_{k \text{ times}} \Big)$$

Proposition 4.6 (i) *Consider the monoid morphism $\varphi_n : \mathbb{N}^n \times \mathbb{N}^n \to \mathbb{Z}^n$ defined in Proposition 4.1. Then, for any $A \subseteq \mathbb{Z}^n$, $card(\varphi_n^{-1}(A)) \geq card(A)$.*

(ii) *φ_n can be lifted to a Kleene algebra morphism – which we denote φ_n too. It is the morphism $\varphi_n : \mathcal{P}(\mathbb{N}^n \times \mathbb{N}^n) \to \mathcal{P}(\mathbb{Z})$ defined by $\varphi_n(A) = \big\{\varphi_n(\overline{a}) \mid \overline{a} \in A\big\}$.*

(iii) *For any two sets $A, B \subseteq \mathbb{Z}^n$,*

$$card(A + B) \geq card(A) + card(B) \text{ and } card(A^*) \geq card(A).$$

(iv) *For any finite set $A \subseteq \mathbb{N}^n \times \mathbb{N}^n$, $card(\varphi_n(A^{\circledast})) < \infty$ if and only if $\varphi_n(A) = \{0_n\}$, and in this case we have*

$$A^{\circledast} = \{\mathbf{0}_{2n}\} \cup \bigcup_{k\leq n} A^{k\ominus}$$

As a corollary, $A^{\circledast}$ is finite.

Proof. We will only prove the last property. The left-to-right implication is straightforward, since $\varphi_n(A) \neq \{0_n\}$ implies that, for any $\overline{a} \in \varphi_n(A)$, $k\overline{a} = (ka_1, \ldots ka_n) \in \varphi_n(A^{\circledast})$. For the reverse implication, suppose $\varphi_n(A) = \{\mathbf{0}_n\}$. Hence, for any $(\overline{a}, \overline{a}') \in A$, $\overline{a} = \overline{a}'$.

Let us observe that, for any $\overline{a}, \overline{b} \in \mathbb{N}^n$,

$$\big(\overline{a}, \overline{a}\big) \ominus \big(\overline{b}, \overline{b}\big) = \big(\overline{c}, \overline{c}\big) \text{ where } \forall i \in [n], \overline{c}_i = \max\big(\overline{a}_i, \overline{b}_i\big) \tag{7}$$

It then follows that

$$\big(\overline{a},\overline{a}\big) \ominus \big(\overline{b},\overline{b}\big) = \big(\overline{b},\overline{b}\big) \ominus \big(\overline{a},\overline{a}\big) \tag{8}$$
$$\big(\overline{a},\overline{a}\big) \ominus \big(\overline{a},\overline{a}\big) = \big(\overline{a},\overline{a}\big) \tag{9}$$

Then, for any $\overline{c} \in A^{k\ominus}$ with $k > card(A)$, we have that

$$\overline{c} = \overline{c}_1 \ominus \overline{c}_2 \ominus \ldots \ominus \overline{c}_k. \tag{10}$$

But since A has less than k elements, two c_is must be equal – say, $c_i = c_j$, for $1 \leq i < j \leq k$. On the other hand, by identity (8) above, we may rearrange the decomposition of c from (10) such that c_i and c_j occur one next to the other. But then identity (9) assures us that $c_i \ominus c_j = c_i$, hence c is decomposed into $k-1$ elements from A.

It then follows, by induction on k, that c can be decomposed into at most $card(A)$ elements. But we may show that we can reduce the number of elements in this decomposition to at most n elements due to the identity 7: consider a decomposition

$$\overline{d} = \overline{d}_1 \ominus \ldots \ominus \overline{d}_l \text{ with } l > n$$

Then, by means of identity 7, for each $1 \leq i \leq n$, $\overline{d}_i = \max\big\{(\overline{d}_j)_i \mid 1 \leq j \leq n\big\}$. Therefore we may define the sequence $(j_i)_{i\in[n]}$ such that $\overline{d}_i = (\overline{d}_{j_i})_i$ for all $i \in [n]$. But this means that only at most n of the d_i's really "participate" in the decomposition – or, in other words, $\overline{d} = \overline{d}_{j_1} \ominus \ldots \ominus \overline{d}_{j_n}$, fact which ends our proof. □

Proposition 4.7 *Suppose $card(\varphi_e(E)) \geq 2$ for some expression E over $\Sigma_{\underline{\mathtt{t}}} \cup \Pi$. If E' is a regular expression with colored parentheses which contains E as a subexpression, then E' cannot be balanced.*

Proof. It is sufficient to prove that, for any regular expression with colored parentheses E' which contains E as a subexpression, $card(\varphi_e(E')) \geq 2$ and $card(\mathsf{lr}_e(E')) \geq 2$. The result will then follow by means of Remark 4.4.

The proof of these two inequalities runs by straightforward structural induction on E', using Proposition 4.6. □

By assembling results from Propositions 4.5 and 4.6, we may give the following effective variant of the mapping lr_e:

$$\begin{aligned}
\mathsf{eff}(\xi) &= \big\{(\mathbf{0}_n, \mathbf{0}_n)\big\} \text{ for all } \xi \in \Sigma_{\underline{\mathbf{t}}} \\
\mathsf{eff}(\langle^i) &= \big\{(e_i, \mathbf{0}_n)\big\} \\
\mathsf{eff}(\rangle^i_I) &= \big\{(\mathbf{0}_n, e_i)\big\} \\
\mathsf{eff}(E_1 + E_2) &= \begin{cases} \bot & \text{iff } \mathsf{eff}(E_1) = \bot \text{ or } \mathsf{eff}(E_2) = \bot \\ \bot & \text{iff } \varphi_n(\mathsf{eff}(E_1)) \neq \varphi_n(\mathsf{eff}(E_2)) \\ \mathsf{eff}(E_1) \cup \mathsf{eff}(E_2) & \text{otherwise} \end{cases} \\
\mathsf{eff}(E_1 \cdot E_2) &= \begin{cases} \bot & \text{iff } \mathsf{eff}(E_1) = \bot \text{ or } \mathsf{eff}(E_2) = \bot \\ \bot & \text{iff } \mathit{card}\big(\varphi_n(\mathsf{eff}(E_1) \ominus \mathsf{eff}(E_2))\big) \neq 1 \\ \mathsf{eff}(E_1) \ominus \mathsf{eff}(E_2) & \text{otherwise} \end{cases} \\
\mathsf{eff}(E^*) &= \begin{cases} \bot & \text{iff } \mathsf{eff}(E) = \bot \\ \bot & \text{iff } \varphi_n(\mathsf{eff}(E)) \neq \{\mathbf{0}_n\} \\ \bigcup_{k \leq n} \mathsf{eff}(E)^{k\ominus} & \text{otherwise} \end{cases}
\end{aligned}$$

Since the recursive definition above involves only finite sets, it can be used as an algorithm to compute $\mathsf{eff}(E)$. The relation between $\mathsf{eff}(E)$ and $\mathsf{lr}_e(E)$ is described in the following proposition:

Proposition 4.8 *For each regular expression E over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$,*

$$\mathsf{eff}(E) = \begin{cases} \bot & \textit{iff } \mathit{card}(\mathsf{lr}_e(E)) = \infty \\ \mathsf{lr}_e(E) & \textit{otherwise} \end{cases}$$

The main result of this section is now immediate:

Theorem 4.9 *A regular expression E over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ is balanced iff $\mathsf{eff}(E) = \{(\mathbf{0}_n, \mathbf{0}_n)\}$.*

5 Proof of the inverse inclusion in the Kleene theorem

The idea of this proof is to transform each parenthesis $\langle^i$ into a clock reset, and each parenthesis $\rangle^i_I$ into a clock check. However, the exact identity of the clock which is reset when encountering $\langle^i$ depends on the "nesting" of the parentheses of color i – that is, for each parenthesis $\langle^i$ we need to create several clocks, as many as the *maximal nesting degree* of the parentheses of color i. But we first need to prove that this nesting degree has an upper bound.

Definition 5.1 Given a word $w \in \Sigma_{\underline{\mathbf{t}}} \cup \Pi$ (non-necessarily balanced), the **maximal nesting degree** of parentheses within w is the tuple $\mathsf{Mdeg}(w) = \big(\mathsf{Ml}(w), \mathsf{Mr}(w)\big) \in \mathbb{N}^{2n}$ whose components are:

$$\begin{aligned}
\mathsf{Ml}(w)_i &= \max\big\{l(w_1)_i \mid w = w_1 w_2\big\} \text{ for each } i \in [n] \\
\mathsf{Mr}(w)_i &= \max\big\{r(w_2)_i \mid w = w_1 w_2\big\} \text{ for each } i \in [n]
\end{aligned}$$

Given an expression E over $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$, the **maximal nesting degree** for E is the tuple $\mathsf{Mdeg}(E) \in \mathbb{N}^{2n}$ whose components are

$$\mathsf{Mdeg}(E)_i = \max\bigl\{\mathsf{Mdeg}(w)_i \mid w \in |E|\bigr\} \text{ for each } i \in [n]$$

Remark 5.2 If w is balanced then $\mathsf{Ml}(w) = \mathsf{Mr}(w)$.

Proposition 5.3 *For each balanced regular expression E, there exists a tuple of integers $M \in \mathbb{N}^{2n}$ such that for each generated word $w \in |E|$, $\mathsf{Mdeg}(w) \leq M$.*

Here, the order is the usual extension of $\leq$ from the naturals to tuples of naturals, i.e., for all $\overline{a}, \overline{b} \in \mathbb{N}^{2n}$, $\overline{a} \leq \overline{b}$ iff $a_i \leq b_i$ for all $1 \leq i \leq 2n$.

Proof. By means of Remark 5.2 we only need to prove the existence of an n-uple of integers $M \subseteq \mathbb{N}$ such that $\mathsf{Ml}(w) \leq M$.

We may compute an upper bound for the maximal nesting degree of left parentheses in E, inductively on the structure of E, as follows:

$$\begin{gathered} M(a) = M(\underline{\mathbf{t}}) = \{\mathbf{0}_n\} \text{ for all } a \in \Sigma \\ M(\langle^i) = \{e_i\}, \qquad M(\rangle^i_I) = \{\mathbf{0}_n\} \end{gathered}$$

Then, for each $1 \leq i \leq n$,

$$\begin{aligned} M(E_1 + E_2)_i &= \max\Bigl(M(E_1)_i, M(E_2)_i\Bigr) \\ M(E_1 \cdot E_2)_i &= \max\bigl(\bigl(l(E_1)_i \dot{-} r(E_2)_i\bigr) + M(E_2)_i, M(E_1)_i\bigr) \\ M(E_1^*)_i &= \max\bigl(\bigl(l(E_1)_i \dot{-} r(E_1)_i\bigr) + M(E_1)_i\bigr) \end{aligned}$$

It is then easy to prove that, for each E' subexpression of E, $M(E') \geq \mathsf{Ml}(E')$. $\square$

Proof. [of the Kleene theorem 3.5] Let $\mathcal{A} = (Q, \delta, q_0, Q_f)$ be the finite automaton over the alphabet $\Sigma_{\underline{\mathbf{t}}} \cup \Pi$ which is equivalent with the expression E.

We will add in each state q some information regarding the maximal number of unbalanced left parentheses in words w for which $q_0 \xrightarrow{w} q$ in A – denote it $M(q)$. We then observe that $M(q)$ is always smaller than the above computed $M(E)$ since w is a left factor of an accepted word and therefore its number of unbalanced left parentheses is smaller than $\mathsf{Ml}(E) \leq M(E)$. Hence the quantity of information we add into states is finite.

The formalization of this step is the following: consider first the set of states

$$Q' = Q \times \bigl\{\psi : [n] \to \mathbb{N} \mid \psi(i) \leq M(E)_i\bigr\}$$

The function ψ in a tuple (q, ψ) is the additional memory for each state, and records the number of left parentheses which are unbalanced in any word w for which $q_0 \xrightarrow{w} q$.

The transitions are the following:

(i) If $q \xrightarrow{z} q' \in \delta$ for some $z \in \Sigma_{\underline{\mathbf{t}}}$ then $(q, \psi) \xrightarrow{z} (q', \psi) \in \delta'$.

(ii) If $q \xrightarrow{\langle^i} q' \in \delta$ then $(q, \psi) \xrightarrow{\langle^i} (q', \psi') \in \delta'$ where $\psi'(i) = \psi(i) + 1$ and $\psi'(j) = \psi(j)$ for all $j \neq i$.

(iii) If $q \xrightarrow{\rangle^i_I} q' \in \delta$ then $(q, \psi) \xrightarrow{\rangle^i_I} (q', \psi') \in \delta'$ where $\psi'(i) = \psi(i) - 1$ and $\psi'(j) = \psi(j)$ for all $j \neq i$.

Observe that we employ "minus" rather than "monus" in the last type of transitions. The reason is that we cannot arrive in a state q with a word containing more right parentheses than left parentheses for a certain color, since otherwise our expression would not be balanced.

Denote also ψ_0 the identically null function $\psi_0(i) = 0$ for all $i \in [n]$. The resulting automaton is then $\mathcal{A}' = \big(Q'', \delta', (q_0, \psi_0), Q_f \times \{\psi_0\}\big)$ where Q'' is the reachable part of Q'. The fact that, for each $i \in [n]$, $M(E)_i$ is an upper bound for the maximal nesting degree for color i implies that the languages of $\mathcal{A}$ and $\mathcal{A}'$ are equal.

Finally, we transform the automaton $\mathcal{A}'$ into a timed automaton by replacing each left parenthesis with a reset of some clock and each right parenthesis with a clock check. The number of clocks that correspond to color i is bounded by $M(E)_i$, and the idea is that, if we are in state (q, ψ) and must do a $\langle^i$ transition in $\mathcal{A}'$, we reset then the $(\psi(i) + 1)$–th clock for color i. We will also need a supplementary clock for imposing that no time elapses between two non-$\underline{\mathbf{t}}$-transitions.

Formally, the resulting timed automaton $\overline{\mathcal{A}}$ has Q'' as its state set, and the following set of clocks:

$$\mathcal{X} = \big\{x^i_j \mid i \in [n], j \in [M(E)_i]\big\} \cup \{y\}$$

Its transitions are the following:

(i) If $(q, \psi) \xrightarrow{a} (q', \psi) \in \delta'$ then $(q, \psi) \xrightarrow{a, y=0?, y:=0} (q', \psi) \in \overline{\delta}$.

(ii) If $(q, \psi) \xrightarrow{\underline{\mathbf{t}}} (q', \psi) \in \delta$ then $(q, \psi) \xrightarrow{\varepsilon, y:=0} (q', \psi) \in \overline{\delta}$.

(iii) If $(q, \psi) \xrightarrow{\langle^i} (q', \psi') \in \delta$ then $(q, \psi) \xrightarrow{\varepsilon,\ y=0?, y:=0, x^i_{\psi'(i)}:=0} (q', \psi') \in \overline{\delta}$.

(iv) If $(q, \psi) \xrightarrow{\rangle^i_I} (q', \psi') \in \delta$ then $(q, \psi) \xrightarrow{\varepsilon,\ (x^i_{\psi'(i)} \in I) \wedge (y=0)?, y:=0} (q', \psi') \in \overline{\delta}$.

The sets of initial and final states are the same as in $\mathcal{A}'$. $\square$

6 Conclusion

The main contribution of this paper is a new class of regular expressions capable to describe all timed regular languages. This formalism, unlike its predecessors, does not use strange operations nor explicit clocks in the expressions, at the price of having a two-stage non-compositional semantics. The problem of practical methods to specify timed behaviors, which could be based on existing or new formalisms, needs further investigation (see [15] for a preliminary discussion).

Another (more technical) contribution of this paper is theorem 2.1 based on a convexity-based "normalization" of timed automata. We have found this transformation very useful. A similar transformation has been applied directly to expressions in [4].

References

[1] Alur, R. and D. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), pp. 183–235.

[2] Annichini, A., E. Asarin and A. Bouajjani, *Symbolic techniques for parametric reasoning about counter and clock systems*, in: *Proceedings of CAV'00*, LNCS **1855**, 2000, pp. 419–434.

[3] Asarin, E., P. Caspi and O. Maler, *A Kleene theorem for timed automata*, in: *Proceedings of LICS'97*, 1997, pp. 160–171.

[4] Asarin, E., P. Caspi and O. Maler, *Timed regular expressions*, Journal of the ACM **49** (2002), pp. 172–206.

[5] Bouyer, P. and A. Petit, *Decomposition and composition of timed automata*, in: *Proceedings of ICALP'99*, LNCS **1644**, 1999, pp. 210–219.

[6] Bouyer, P. and A. Petit, *A Kleene/Büchi-like theorem for clock languages*, Journal of Automata, Languages and Combinatorics **7** (2002), pp. 167–186.

[7] Bouyer, P., A. Petit and D. Thérien, *An algebraic characterization of data and timed languages*, in: *Proceedings of CONCUR'2001*, LNCS **2154**, 2001, pp. 248–261.

[8] Conway, J. H., "Regular Algebra and Finite Machines," Chapman and Hall, 1971.

[9] Daws, C., "Méthodes d'analyse de systèmes temporisés: de la théorie à la pratique," Ph.D. thesis, Institut National Polytechnique de Grenoble, France (1998).

[10] Daws, C. and S. Yovine, *Reducing the number of clock variables of timed automata*, in: *Proceedings of RTSS'96* (1996).

[11] Dima, C., "An algebraic theory of real-time formal languages," Ph.D. thesis, Université Joseph Fourier, Grenoble, France (2001).

[12] Dima, C., *Timed regular expressions with colored parentheses* (2002), submitted.

[13] Herrmann, P., *Renaming is necessary in timed regular expressions*, in: *Proceedings of FST&TCS'99*, LNCS **1738**, 1999, pp. 47–59.

[14] Hopcroft, J. and J. Ullman, “Introduction to Automata Theory, Languages and Computation,” Addison-Wesley/Narosa Publishing House, 1992.

[15] Moy, M., “Spécifications des comportements temporisés,” Master’s thesis, Institut National Polytechnique de Grenoble (2002).

[16] Sheng Yu, “Regular Languages,” Handbook of Formal Languages **1**, Springer Verlag, 1997.