Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 106 (2004) 315–333

www.elsevier.com/locate/entcs

On Equivalences for a Class of Timed Regular Expressions

Riccardo Pucella

Department of Computer Science
Cornell University
Ithaca, NY 14853 USA
mailto: riccardo@cs. cornell. edu

Abstract

Timed regular expressions are an extension of regular expressions that capture a notion of time. Roughly speaking, timed regular expressions can be used to represent timed sequences of events, with new operators to control the duration of those sequences. These timed regular expressions correspond to a form of timed automaton equipped with clocks, of the kind introduced by Alur and Dill. We develop a coalgebraic treatment of such timed regular expressions, along the lines of the coalgebraic treatment of regular expressions based on deterministic automata. This yields a coinductive proof principle, that can be used to establish equivalence of a class of timed regular expressions.

Keywords: TImed regular expression, timed automata, deterministic automata, coinduction.

1 Introduction

Timed automata, introduced by Alur and Dill [1], have become a popular framework for modeling and specifying the behaviour of real-time systems. Their naturalness, in some sense, is reflected by the fact that certain classes of timed automata are equivalent to various real-time logics that have been used to reason about real-time systems (see Wilke [19] and references therein). Perhaps more importantly, timed automata are a generalization of classical automata, and this leads to the possibility that well-known results from classical automata theory can be extended and adapted to the theory of timed automata [17].

doi:10.1016/j.entcs.2004.02.042

In analogy with classical automata theory, the behaviour of timed automata can also be expressed using an extension of regular expressions, called timed regular expressions. Roughly speaking, timed regular expressions denote sets of timed words, which not only record sequences of event occurrences, but also the time elapsed between these occurrences. However, while regular expressions exactly capture the languages recognized by classical automata, the corresponding relationship for timed automata is more complex. The straightforward extension of regular expressions to timed regular expressions (via an operator that constrains the duration of the timed words expressed by a timed regular expression) does not fully capture the languages recognized by timed automata. There is still disagreement about the "right" extension to timed regular expression that can fully match the expressive power of timed automata: one proposal describe an extension with an intersection operator and a renaming operator [3], while another advocates a whole family of operators to constrain the duration of timed words [4,9].

In this paper, we initiate a study of the properties of timed regular expressions. We focus on an uncontroversial subclass of timed regular expressions, that is, timed regular expressions with a single operator to constrain the duration of time words, and investigate the problem of determining when two such timed regular expressions are equivalent, that is, when they denote the same language. This problem is nontrivial, since in contrast with the regular expressions case [16,13], there is no known sound and complete axiomatization for timed regular expressions.

Our approach follows that of Rutten [14], who develops a coalgebraic theory of classical automata. One of the consequences of this coalgebraic treatment is *coinduction*, a proof technique for demonstrating the equivalence of classical languages, which can be lifted to a procedure that decides the equivalence of regular expressions. We extend the work of Rutten to the case of timed regular expressions. More precisely, we give a coalgebraic treatment of timed automata, derive a coinduction proof principle, and show that this can be used to decide the equivalence of a class of timed regular expressions.

This paper is structured as follows. In the next section, we review the formulation of timed words and timed languages that we use. In Section 3, we define timed regular expressions, an extension of regular expressions with an operator that restricts the duration of timed words. In Section 4, we define a form of deterministic timed automaton that can accept timed languages, and show in Section 5 that these automata form a category DTA with an object that is almost final consisting of a timed automaton whose states are timed languages. This gives rise to a coinduction proof principle for timed languages equality. In Section 6, we use this coinduction proof principle to

derive an algorithm for establishing equivalence of timed regular expressions. We conclude in Section 7. For reasons of space, we leave the proofs of our results for the full paper.

2 Timed Words and Timed Languages

Classical language theory [11] studies words over an alphabet Σ. One interpretation of such words is that they represent sequences of "events" occuring consecutively. From this point of view, a timed word (also called a time-event sequence) is a word that records not only the events, but also the time elapsed between these, with respect to some unit of time. For example, $ab \cdot 3 \cdot c$ represents the events a and b occuring together, followed by a delay of 3 time units, followed by the event c.

To formally define timed words, we take the approach of Asarin, Caspi, and Maler [3]. Recall that classical words over Σ form the free monoid $(\Sigma^*, \cdot^*, \epsilon^*)$ generated by Σ, where Σ^* is the set of all finite sequences of elements of Σ, $\cdot^*$ is concatenation (we will from now on write ab rather than $a \cdot^* b$ for the concatenation of elements in Σ^*), and ϵ^* is the empty word. To define timed words, we take the combination of the monoid $(\Sigma^*, \cdot^*, \epsilon^*)$ with the monoid $(\mathbb{R}_{\geq 0}, +, 0)$ of nonnegative real numbers, used to represent time. More precisely, we define the monoid $\mathcal{T}(\Sigma)$ of timed words over Σ as the quotient of the free monoid over $\Sigma^* \cup \mathbb{R}_{\geq 0}$ (with concatenation $\cdot$ and empty word ϵ) via the congruence relation $\sim$:

$$\sigma_1 \cdot \sigma_2 \sim \sigma_1\sigma_2 \text{ if } \sigma_1, \sigma_2 \in \Sigma^*$$
$$t_1 \cdot t_2 \sim t_1 + t_2 \text{ if } t_1, t_2 \in \mathbb{R}_{\geq 0}$$
$$\epsilon^* \sim \epsilon$$
$$0 \sim \epsilon.$$

Intuitively, these congruence rules allow us to replace adjacent elements from the same monoid in a sequence in $(\Sigma^* \cup \mathbb{R}_{\geq 0})^*$ by a single element, and to get rid of dummy identity elements. The upshot of this quotienting is that every timed word can be written in canonical form, $t_1 \cdot a_1 \cdot t_2 \cdot a_2 \cdots$, that is, as an alternation of elements of Σ^* and $\mathbb{R}_{\geq 0}$. We will in fact generally be concerned only with words where the last element of the word (when in canonical form) is an element of Σ. (Intuitively, we forget about time delays when they occur at the end of the word.) We call these words *normal timed words*. If $\sigma = t_1 \cdot a_1 \cdots t_n \cdot a_n \cdot t_{n+1}$ is a timed word, then $t_1 \cdot a_1 \cdots t_n \cdot a_n$ is the normal timed word corresponding to σ. A *timed language* is a set of timed words, and a *normal timed language* is a set of normal timed words. If L is a timed language, then the language made up of all the normal timed words

corresponding to the timed words in L is called the normal timed language corresponding to L.

Example 2.1 The timed language $\{t_1 \cdot a \cdot t_2 \cdot b \cdot t_3 \cdot c \mid t_2 + t_3 \leq 10\}$ is the set of all timed words where there is at most 10 units separating the occurrence of the a and c events, with an intervening b event. The timed language $\{t_1 \cdot a \cdot t_2 \cdot b \cdot t_3 \cdot c \mid t_1 + t_2 \leq 10, t_2 + t_3 \leq 10\}$ is the set of all timed words where the events a and b occur with 10 time units of the start, and moreover the events a and c occur within 10 time units of each other, with an intervening b event.

Remark 2.2 Alternative notations for timed strings can be found in the literature. In the original work of Alur and Dill [1], a timed word over Σ is a pair (σ, τ) where σ is a word in Σ^*, and τ is an infinite sequence of time values in $\mathbb{R}_{\geq 0}$, assumed to be monotone and unbounded. Intuitively, if each element σ_i of σ is considered an event, then the corresponding τ_i represent the time of occurrence of σ_i. It should be clear that the two representations are equivalent, in the sense that we can freely convert between the two. The notation we use has the advantage that word concatenation is easier to define, and behaves somehow more naturally. This makes it easier to develop the theory in this paper. A different notation is used by Asarin, Caspi, and Maler [2]. They define a *signal* over Σ to be a string of the form $a_1^{t_1} \ldots a_k^{t_k}$, where $a_i^{t_i}$ is meant to represent the event a_i occuring and lasting for time t_i. In their representation, an event lasts until the next event occurs. If we identify our events with the instantaneous first occurrence of an event in the signal notation, there is again an immediate correspondence between the two notations.

Given a timed word $\sigma \in \mathcal{T}(\Sigma)$, the *duration* of σ, that is, the amount of time spent in σ, can be defined by projection. The function $\lambda : \mathcal{T}(\Sigma) \longrightarrow \mathbb{R}_{\geq 0}$ is obtained by mapping elements of Σ^* to 0, and considering the result as an element of the monoid $(\mathbb{R}_{\geq 0}, +, 0)$. For example, $\lambda(ab \cdot 3 \cdot c) = 0 \cdot 3 \cdot 0 = 0 + 3 + 0 = 3$.

The concatenation $L_1 \cdot L_2$ of two languages is the language $\{\sigma_1 \cdot \sigma_2 \mid \sigma_1 \in L_1, \sigma_2 \in L_2\}$. Observe that $\varnothing \cdot L = L \cdot \varnothing = \varnothing$. We write L^n for the n-fold concatenation of L with itself, that is, $L \cdot \ldots \cdot L$ (n times). As usual, $L^0 = \{\epsilon\}$. Finally, we write L^* for $\cup_{n=0}^{\infty} L^n$. An important operation on languages is that of taking the *derivative*, a notion adapted from [6]. Given a timed word σ, the derivative D_σ of a timed language L is given by $D_\sigma(L) = \{\sigma' \mid \sigma \cdot \sigma' \in L\}$. We will be mostly interested in derivatives of the form $D_{t \cdot a}(L)$ for $t \in \mathbb{R}_{\geq 0}$ and $a \in \Sigma$.

3 Timed Regular Expressions

Timed regular expressions can be used to denote certain class of languages, without referring to explicit constraints over the individual times. A common notation starts with the following core:

$$e ::= 0 \mid 1 \mid a \mid e_1 e_2 \mid e_1 + e_2 \mid e^* \mid \langle e \rangle_I$$

where a represents an arbitrary element of Σ, and I is an interval of the form $[l,u]$, $[l,u)$, $(l,u]$, or (l,u) (with $l,u \in \mathbb{R}_{\geq 0}$ and $l \leq u$), or of the form $[l,\infty)$ or (l,∞) (with $l \in \mathbb{R}_{\geq 0}$). Regular expressions operators maintain their intuitive interpretation: 0 is the null regular expression, 1 is the regular expression representing the empty word, a represent the event $a \in \Sigma$, e_1 e_2 is the concatenation of the events in e_1 and e_2, $e_1 + e_2$ is the disjunction of the events in e_1 and those in e_2, and e^* is the iteration of the events in e an arbitrary but finite number of times. These operators do not impose any restriction on the timing of the events. Restrictions are captured using the new construct $\langle e \rangle_I$. Intuitively, $\langle e \rangle_I$ indicates that the events specified by e are restricted to a duration in the interval I. For example, $\langle ab \rangle_{[0,10]} c$ represents the events a and b occuring within 10 time units, followed by the event c.

Remark 3.1 We should note that we allow the bounds on intervals I to be real-valued, whereas in the literature they are taken to be integer-valued. The added expressiveness of allowing real-valued bounds will be important later in this section to define a useful notion of derivative for timed regular expressions.

In analogy with regular expressions, we can associate with every timed regular expression e a normal timed language $[\![e]\!]$ as follows:

$$\begin{aligned}
[\![0]\!] &= \varnothing \\
[\![1]\!] &= \{\epsilon\} \\
[\![a]\!] &= \{t \cdot a \mid t \in \mathbb{R}_{\geq 0}\} \\
[\![e_1 e_2]\!] &= [\![e_1]\!] \cdot [\![e_2]\!] \\
[\![e_1 + e_2]\!] &= [\![e_1]\!] \cup [\![e_2]\!] \\
[\![e^*]\!] &= [\![e]\!]^* \\
[\![\langle e \rangle_I]\!] &= [\![e]\!] \cap \{\sigma \mid \lambda(\sigma) \in I\}.
\end{aligned}$$

The definition is standard, except for the language associated with an event a, which is the language representing an arbitrary time delay before the occurrence of a; this captures the fact that there is a priori no timing restrictions on events.

Example 3.2 Returning to the example above, the timed language associated with the timed regular expression $\langle ab\rangle_{[0,10]}c$ is $\{t_1 \cdot a \cdot t_2 \cdot b \cdot t_3 \cdot c \mid t_1 + t_2 \leq 10\}$. To get a feel for the $\langle e\rangle_I$ operator, compare the following languages:

$$[\![\langle a\rangle_{[0,1)}\langle a\rangle_{[0,1)}]\!] = \{t_1 \cdot a \cdot t_2 \cdot a \mid t_1 < 1, t_2 < 1\}$$
$$[\![\langle aa\rangle_{[0,2)}]\!] = \{t_1 \cdot a \cdot t_2 \cdot a \mid t_1 + t_2 < 2\}$$
$$[\![\langle\langle a\rangle_{[0,1)}a\rangle_{[0,2)}]\!] = \{t_1 \cdot a \cdot t_2 \cdot a \mid t_1 < 1, t_1 + t_2 < 2\}$$
$$[\![\langle a\langle a\rangle_{[0,1)}\rangle_{[0,2)}]\!] = \{t_1 \cdot a \cdot t_2 \cdot a \mid t_2 < 1, t_1 + t_2 < 2\}.$$

In particular, note that all of these languages are distinct. For instance, $[\![\langle aa\rangle_{[0,2)}]\!]$ contains the word $0.5 \cdot a \cdot 1.2 \cdot a$ which is not in $[\![\langle a\rangle_{[0,1)}\langle a\rangle_{[0,1)}]\!]$ nor in $[\![\langle a\langle a\rangle_{[0,1)}\rangle_{[0,2)}]\!]$, as well as the word $1.2 \cdot a \cdot 0.5 \cdot a$, which is not in $[\![\langle\langle a\rangle_{[0,1)}a\rangle_{[0,2)}]\!]$. Compare the situation with the following languages:

$$[\![\langle aa\rangle_{[0,2)}]\!] = \{t_1 \cdot a \cdot t_2 \cdot a \mid t_1 + t_2 < 2\}$$
$$[\![\langle\langle a\rangle_{[0,2)}a\rangle_{[0,2)}]\!] = \{t_1 \cdot a \cdot t_2 \cdot a \mid t_1 < 2, t_1 + t_2 < 2\}$$
$$[\![\langle a\langle a\rangle_{[0,2)}\rangle_{[0,2)}]\!] = \{t_1 \cdot a \cdot t_2 \cdot a \mid t_2 < 2, t_1 + t_2 < 2\}.$$

All these languages are equal.

While the notion of timed regular expressions we give above captures many forms of timed languages, they are still fairly restricted. For example, they cannot capture the language $\{t_1 \cdot a \cdot t_2 \cdot b \cdot t_3 \cdot c \mid t_1 + t_2 \leq 10, t_2 + t_3 \leq 10\}$ from Example 2.1. Why is this an issue at all? Recall that in the classical theory of languages, the languages expressible by regular expressions correspond exactly to the languages recognized by finite state automata. One can ask whether timed regular expressions can capture the languages recognized by the timed automata defined by Alur and Dill [1], the motivation for introducing timed regular expressions in the first place. (We shall define timed automata in the next section, albeit with a slightly different aim.) It turns out that the answer is no: the language $\{t_1 a \cdot t_2 \cdot b \cdot t_3 \cdot c \mid t_1 + t_2 \leq 10, t_2 + t_3 \leq 10\}$ is recognizable using a fairly simple timed automaton. Researchers have endeavoured to capture the languages recognized by timed automata. Two approaches have been proposed. [1] The first, described by Asarin, Caspi, and Maler [2,3], extends the timed regular expressions defined above with an intersection operator and a renaming operator. They prove that these extended timed regular expressions exactly capture timed languages recognized by timed automata. [2] Unfortunately, the renaming operator is rather heavy-handed. An alternative approach, described by Asarin and Dima [4] and Dima [9], avoids the need for

[1] Bouyer and Petit [5] introduce an alternative form of timed regular expression, different in spirit than the one in this section, that are very close to timed automata themselves.
[2] Herrmann [10] first recognized that renaming was in fact necessary to establish the result.

either an intersection or a renaming operator, and uses a family of $\langle\ \rangle_I$ operators, each corresponding essentially to a different constraint on the times in timed word. The difficulty of this approach consists of the fact that well-formedness of timed regular expressions becomes difficult to establish, since the different $\langle\ \rangle_I$ operators need not nest.

Despite the fact that the timed regular expressions we introduce are not expressive enough to capture all languages recognized by the deterministic automata of Alur and Dill [1], we focus on these core timed regular expressions for the remainder of this paper.

In the last section, we gave a definition of derivative for timed languages. We now introduce a syntactic operation corresponding to taking the derivative $D_{t\cdot a}$ on languages, which we write $\hat{D}_{t,a}$. This can be defined purely syntactically. First, for an interval I and $t \in \mathbb{R}_{\geq 0}$, define $I - t = \{x \mid x + t \in I\} \cap \mathbb{R}_{\geq 0}$. Intuitively, $I - t$ consists of "shifting" the interval I to the left by t.

$$
\begin{aligned}
\hat{D}_{t,a}(0) &= 0 \\
\hat{D}_{t,a}(1) &= 0 \\
\hat{D}_{t,a}(a) &= 1 \\
\hat{D}_{t,a}(b) &= 0 \\
\hat{D}_{t,a}(e_1 e_2) &= \hat{D}_{t,a}(e_1) e_2 + \hat{\epsilon}(e_1) \hat{D}_{t,a}(e_2) \\
\hat{D}_{t,a}(e_1 + e_2) &= \hat{D}_{t,a}(e_1) + \hat{D}_{t,a}(e_2) \\
\hat{D}_{t,a}(e^*) &= \hat{D}_{t,a}(e) e^* \\
\hat{D}_{t,a}(\langle e \rangle_I) &= \begin{cases} \langle \hat{D}_{t,a}(e) \rangle_{I-t} & \text{if } I - t \neq \varnothing \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

This definition relies on an operation $\hat{\epsilon}$ that syntactically checks whether an expression can denote the empty word. Again, this can be defined inductively on the structure of timed regular expressions:

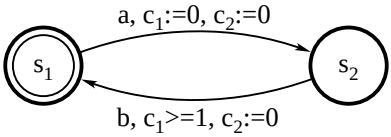

Fig. 1. A timed automaton

$$
\begin{aligned}
\hat{\epsilon}(0) &= 0 \\
\hat{\epsilon}(1) &= 1 \\
\hat{\epsilon}(a) &= 0 \\
\hat{\epsilon}(e_1 e_2) &= \begin{cases} 1 \text{ if } \hat{\epsilon}(e_1) = \hat{\epsilon}(e_2) = 1 \\ 0 \text{ otherwise} \end{cases} \\
\hat{\epsilon}(e_1 + e_2) &= \begin{cases} 0 \text{ if } \hat{\epsilon}(e_1) = \hat{\epsilon}(e_2) = 0 \\ 1 \text{ otherwise} \end{cases} \\
\hat{\epsilon}(e^*) &= 1 \\
\hat{\epsilon}(\langle e \rangle_I) &= \hat{\epsilon}(e).
\end{aligned}
$$

Clearly, $\hat{\epsilon}(e)$ is always the expression 0 or 1. We can verify that the operator $\hat{D}_{t,a}$ indeed captures the $D_{t \cdot a}$ operator defined on timed languages.

Proposition 3.3 *For all timed regular expressions e, $t \in \mathbb{R}_{\geq 0}$, and $a \in \Sigma$, we have $\hat{\epsilon}(e) = 1$ if and only if $\epsilon \in [\![e]\!]$, and $[\![\hat{D}_{t,a}(e)]\!] = D_{t \cdot a}([\![e]\!])$.*

4 Deterministic Timed Automata

The notion of a timed automaton introduced by Alur and Dill [1] extends that of a classical automaton. As in the classical case, a timed automaton is made up of states, and transitions between states. The difference is that timed automata rely on *clocks* to keep track of time. Clocks are initialized at 0, and they increase monotonically, at the same rate, while the automaton is in any particular state. Transitions are labelled by elements of an alphabet Σ, as usual, but are also potentially guarded by constraints on clocks. For instance, we can specify that a transition from a state can only occur if the value of clock c_1 is between 1 and 2. Moreover, upon a transition, clocks can be updated. Having multiple clocks that can be independently updated means that they can be used to measure the elapsed time between different transitions.

Example 4.1 Consider the automaton in Figure 1, with two states s_1 and s_2. If we start in state s_1 with clocks c_1 and c_2 initialized to 0, then the automaton remains in state s_1 for some nondeterministic amount of time t_1, incrementing the clocks c_1 and c_2 by t_1. After an a transition to state s_2, both clocks c_1 and c_2 are reset to 0. The automaton remains in state s_2 for some nondeterministic amount of time t_2, incrementing the clocks c_1 and c_2 by t_2. When clock $c_1 \geq 1$, the automaton can perform a b transition to state s_1, which resets clocks c_2 to 0. This process can be performed an arbitrary number of times. It is not hard to see that the timed language recognized by the state s_1 of this automaton is $\{t_1 \cdot a \cdot t_2 \cdot b \cdot \ldots \cdot t_{2n-1} \cdot a \cdot t_{2n} \cdot b \mid n \in \mathbb{N}, t_{2i} \geq 1, i \in [1, \ldots, n]\}$.

We formalize this kind of automaton in a general way, in fact, much more general than the definition given by Alur and Dill [1]. As in the last section, let Σ be a finite alphabet. If C is a set of clocks, a *clock valuation* $\mathbf{v}$ on C gives, for every clock in C, the time reading of the clock. Formally, $\mathbf{v}$ is a function from C to $\mathbb{R}_{\geq 0}$, that is, $\mathbf{v}(c) \geq 0$ for all $c \in C$, where $\mathbf{v}(c)$ is the time value of clock c. Let $\mathbf{V}(C)$ be the set of all clock valuations on C. Let $\mathbf{0}$ denote the zero valuation, that is, the clock valuation with $\mathbf{0}(c) = 0$ for all c. If $\mathbf{v}$ is a clock valuation and t is a real number in $\mathbb{R}_{\geq 0}$, then $\mathbf{v} + t$ is the valuation which is just like $\mathbf{v}$ except that t is added to all the components, that is, $(\mathbf{v}+t)(c) = \mathbf{v}(c)+t$ for all clocks $c \in C$. If $\mathbf{v}$ is a clock valuation, then $\mathbf{v}[c \mapsto t]$ is the clock valuation $\mathbf{v}'$ which is just like $\mathbf{v}$ except that $\mathbf{v}'(c) = t$. We also write $\mathbf{v} \geq t$ if $\mathbf{v}(c) \geq t$ for all $c \in C$.

We define a *deterministic timed automaton* over Σ to be a tuple $M = (S, C, o, \delta)$, where S is a set of states, C is a set of clocks, o is a function that specifies whether a state is accepting ($o(s) = 1$) or not ($o(s) = 0$), and δ is the transition relation of the automaton. For a set S of states, let $T(S, C) = S \times \mathbf{V}(C)$ be the set of *timed states*. Thus, a timed state has the form $(s, \mathbf{v})$, where s is a state, and $\mathbf{v}$ is a clock valuation on C. The transition relation δ is such that for all input symbols a and for all timed states $(s, \mathbf{v})$, we have $\delta((s, \mathbf{v}), a) \in S \times T(S, C)$. Intuitively, $\delta((s, \mathbf{v}), a) = (s', \mathbf{v}')$ indicates that from a timed state $(s, \mathbf{v})$, the automaton can make a move a leading to a new state s' and a new clock valuation $\mathbf{v}'$.

Example 4.2 The automaton of Example 4.1 (and Figure 1) can be captured in our framework as the automaton (S, C, o, δ) where:

- $S = \{s_1, s_2, s_{sink}\}$
- $C = \{c_1, c_2\}$
- $o(s_1) = 1$
  $o(s_2) = 0$

$o(s_{sink}) = 0$

- $\delta((s_1, \mathbf{v}), x) = \begin{cases} (s_2, \mathbf{0}) & \text{if } x = a \\ (s_{sink}, \mathbf{v}) & \text{otherwise} \end{cases}$

  $\delta((s_2, \mathbf{v}), x) = \begin{cases} (s_1, \mathbf{v}[c_2 \mapsto 0]) & \text{if } x = b \text{ and } \mathbf{v}(c_1) \geq 1 \\ (s_{sink}, \mathbf{v}) & \text{otherwise} \end{cases}$

  $\delta((s_{sink}, \mathbf{v}), x) = (s_{sink}, \mathbf{v})$.

The "sink" state s_{sink} is implicit in most automata descriptions.

Remark 4.3 As we mentioned, these automata are more general than the timed automata defined by Alur and Dill [1]. Here are the main differences. Alur and Dill restrict their automata to a finite set of states, and furthermore specify a start state for each timed automaton. More importantly, while we do not impose any restriction on the transitions δ as far as clock valuations are concerned, the transitions in the timed automata of Alur and Dill are required to be guarded by boolean combinations of constraints of the form $x_i \in I$, for I an interval bounded by integers (such as $[0, 1]$, $[4, 8)$, or $(10, \infty)$). This generality of our presentation is required to be able to define the final automaton in Section 5. Finally, the only clock update allowed by Alur and Dill is clock reset: a set of clocks can be reset upon a transition, leaving other clocks unchanged.

We associate with every timed state $(s, \mathbf{v})$ of an automaton M the normal timed language $L_M(s, \mathbf{v})$ recognized by that timed state. We proceed as follows. A timed word $\sigma = t_1 \cdot a_1 \cdot \ldots \cdot t_n \cdot a_n \cdot t_{n+1}$ is recognized by a timed state $(s, \mathbf{v})$ if either:

(1) $\sigma = t$ and $o(s) = 1$, or

(2) if $\sigma = t \cdot a \cdot \sigma'$, and $\delta((s, \mathbf{v} + t), a) = (s', \mathbf{v}')$ with σ' recognized by $(s', \mathbf{v}')$.

Let $L_M(s, \mathbf{v})$ be the set of normal timed words recognized by M at the timed state $(s, \mathbf{v})$. It is sufficient to restrict oneself to normal timed words, as the following proposition shows.

Proposition 4.4 *The timed word σ is recognized by M at the timed state $(s, \mathbf{v})$ if and only if the normal timed word corresponding to σ is recognized by M at the timed state $(s, \mathbf{v})$.*

A *bisimulation* between two timed automata $M = (S, C, o, \delta)$ and $M' = (S', C', o', \delta')$ is a relation $R \subseteq T(S, C) \times T(S', C')$ such that:

(1) for all $(s, \mathbf{v})R(s', \mathbf{v}')$, we have $o(s) = o(s')$;

(2) for all $(s, \mathbf{v})R(s', \mathbf{v}')$ and for all $t \in \mathbb{R}_{\geq 0}$, we have $(s, \mathbf{v}+t)R(s', \mathbf{v}+t)$;

(3) for all $(s, \mathbf{v})R(s', \mathbf{v}')$ and for all $a \in \Sigma$, we have $\delta((s, \mathbf{v}), a)R\delta'((s', \mathbf{v}'), a)$.

A bisimulation between M and itself is called a bisimulation on M. Two timed states $(s, \mathbf{v})$ and $(s', \mathbf{v}')$ are said to be *bisimilar*, denoted by $(s, \mathbf{v}) \sim (s', \mathbf{v}')$, if there exists a bisimulation R such that $(s, \mathbf{v})R(s', \mathbf{v}')$. The relation $\sim$ is the union of all bisimulations, and thus is the greatest bisimulation. For us, the property of interest is that bisimilar states recognize the same language.

Proposition 4.5 *If s is a state of M and s' is a state of M' with $(s, \mathbf{v}) \sim (s', \mathbf{v}')$, then $L_M(s, \mathbf{v}) = L_{M'}(s', \mathbf{v}')$.*

5 The Category DTA

The timed automata defined in the last section can be given a categorical structure. We define the category DTA of deterministic timed automata (over a fixed Σ) by taking as objects deterministic timed automata as above. Given automata $M = (S, C, o, \delta)$ and $M' = (S', C', o', \delta')$, a morphism $f : M \longrightarrow M'$ is a function $f : T(S, C) \longrightarrow T(S', C')$ such that:

(1) for all $(s, \mathbf{v}) \in T(S, C)$, if $f(s, \mathbf{v}) = (s', \mathbf{v}')$, then $o(s) = o'(s')$;

(2) for all $(s, \mathbf{v}) \in T(S, C)$ and $t \in \mathbb{R}_{\geq 0}$, if $f(s, \mathbf{v}) = (s', \mathbf{v}')$, then $f(s, \mathbf{v}+t) = (s', \mathbf{v}'+t)$;

(3) for all $(s, \mathbf{v}) \in T(S, C)$ and $a \in \Sigma$, $f(\delta((s, \mathbf{v}), a)) = \delta'(f(s, \mathbf{v}), a)$.

With morphism composition defined as expected, it is straightforward to check that DTA is indeed a category.

An important property of morphisms is that they preserve the language recognized by timed states. This follows immediately from Proposition 4.5 in conjunction with the following result.

Proposition 5.1 *Let $f : M \longrightarrow M'$ be a morphism. Then $(s, \mathbf{v}) \sim f(s, \mathbf{v})$.*

The main characteristic of the category DTA, for our purposes, is that it possesses an object which is almost, but not quite, final. Intuitively, the set of all timed languages can be given a timed automaton structure. Define the timed automaton $\mathcal{L} = (S_{\mathcal{L}}, C_{\mathcal{L}}, o_{\mathcal{L}}, \delta_{\mathcal{L}})$ as follows:

- $S_{\mathcal{L}}$ is the set of all timed languages;
- $C_{\mathcal{L}}$ is a set with a single clock c;
- $o_{\mathcal{L}}(L) = 1$ if and only if $\epsilon \in L$;
- $\delta_{\mathcal{L}}((L, \mathbf{v}), a) = (D_{\mathbf{v}(c)\cdot a}(L), \mathbf{0})$.

The clock c is reset on every transition, and we use that clock to decide how much time has elapsed between transitions.

We can check that $L_{\mathcal{L}}(K, \mathbf{0}) = K$, that is, the timed language recognized by the timed state $(K, \mathbf{0})$ in $\mathcal{L}$ is K itself. More generally, we have $L_{\mathcal{L}}(K, \mathbf{v}) = D_{\mathbf{v}(c)}(K)$.

Given any timed automaton M, there is a natural morphism from M to $\mathcal{L}$ that maps every timed state of M to the timed language recognized by that timed state in M. For any deterministic timed automaton M, define the morphism $f_F : M \longrightarrow \mathcal{L}$ by taking $f_F(s, \mathbf{v}) = (L_M(s, \mathbf{v}), \mathbf{0})$. While this morphism f_F is not the only morphism from M to $\mathcal{L}$, all other morphisms are closely related.

Proposition 5.2 *For any morphism $g : M \longrightarrow \mathcal{L}$, if $g(s, \mathbf{v}) = (K, \mathbf{v}')$, then $f_F(s, \mathbf{v}) = (D_{\mathbf{v}'(c)}(K), \mathbf{0})$.*

This almost-finality of $\mathcal{L}$ gives rise to the following coinduction proof principle for language equality, in a way which is by now standard [15].

Theorem 5.3 *For two timed languages K and L, if $(K, \mathbf{v}) \sim (L, \mathbf{v}')$ then $D_{\mathbf{v}(c)}(K) = D_{\mathbf{v}'(c)}(L)$.*

In particular, if $(K, \mathbf{0}) \sim (L, \mathbf{0})$, then $K = L$. In other words, to establish the equality of two timed languages, it is sufficient to exhibit a bisimulation between the two languages when viewed as states of the final timed automaton $\mathcal{L}$.

As an application of this principle, we show how to use it to establish the equality of the languages described by the timed regular expressions of Section 3. For example, consider the languages $[\![\langle aa \rangle_{[0,2)}]\!]$ and $[\![\langle \langle a \rangle_{[0,2)} a \rangle_{[0,2)}]\!]$ from Example 3.2. The following relation R is easily seen to be a bisimulation:

$$((([\![\langle aa \rangle_{[0,2-t)}]\!], \mathbf{v}), ([\![\langle \langle a \rangle_{[0,2-t)} a \rangle_{[0,2-t)}]\!], \mathbf{v})) \in R \quad (t \in [0, 2))$$
$$((([\![\langle a \rangle_{[0,2-t)}]\!], \mathbf{v}), ([\![\langle a \rangle_{[0,2-t)}]\!], \mathbf{v})) \in R \quad (t \in [0, 2))$$
$$(([\![1]\!], \mathbf{v}), ([\![1]\!], \mathbf{v})) \in R$$
$$(([\![0]\!], \mathbf{v}), ([\![0]\!], \mathbf{v})) \in R.$$

To establish that R is a bisimulation, we can use the syntactic derivative $\hat{D}$ defined in Section 3. For example, for $\mathbf{v}(c) = 1$, to show that

$$((D_{1 \cdot a}([\![\langle aa \rangle_{[0,2)}]\!]), \mathbf{0}), (D_{1 \cdot a}([\![\langle \langle a \rangle_{[0,2)} a \rangle_{[0,2)}]\!]), \mathbf{0})) \in R,$$

we can check that we have:

$$\begin{aligned} D_{1\cdot a}([\![\langle aa\rangle_{[0,2)}]\!]) &= [\![\hat{D}_{1,a}(\langle aa\rangle_{[0,2)})]\!] \\ &= [\![\langle \hat{D}_{1,a}(aa)\rangle_{[0,1)}]\!] \\ &= [\![\langle \hat{D}_{1,a}(a)a\rangle_{[0,1)}]\!] \\ &= [\![\langle a\rangle_{[0,1)}]\!] \end{aligned}$$

$$\begin{aligned} D_{1\cdot a}([\![\langle\langle a\rangle_{[0,2)}a\rangle_{[0,2)}]\!]) &= [\![\hat{D}_{1,a}(\langle\langle a\rangle_{[0,2)}a\rangle_{[0,2)})]\!] \\ &= [\![\langle \hat{D}_{1,a}(\langle a\rangle_{[0,2)}a\rangle_{[0,1)})]\!] \\ &= [\![\langle\langle \hat{D}_{1,a}(a)\rangle_{[0,1)}a\rangle_{[0,1)}]\!] \\ &= [\![\langle\langle 1\rangle_{[0,1)}a\rangle_{[0,1)}]\!] \\ &= [\![\langle a\rangle_{[0,1)}]\!]. \end{aligned}$$

Similarly for all the other cases. In the next section, we show how to mechanically obtain such bisimulations when the timed languages are described by a specific kind of timed regular expressions.

6 Proving Equivalence of Timed Regular Expressions

Theorem 5.3 gives a proof technique for determining when two timed languages are equal, namely, by exhibiting a bisimulation between the two languages. In this section, we show that this proof technique can be used to prove that two timed regular expressions of a specific kind are equivalent, that is, that they denote the same timed language. Moreover, the technique is complete, in the sense that it can establish the equivalence of any two timed regular expressions that are equivalent. Essentially, we show that given any two equivalent timed regular expressions, we can effectively construct a finite bisimulation relating then, and the construction requires only simple syntactic manipulations of the timed regular expressions.

Clearly, equivalence of timed regular expressions subsumes the problem for regular expressions proper, since every regular expressions is a timed regular expression. There are, however, nontrivial equivalence among timed regular expressions that depend only on the timing operator $\langle e\rangle_I$. Consider the expressions in Example 3.2. Because they denote the same language, $\langle aa\rangle_{[0,2)}$ and $\langle\langle a\rangle_{[0,2)}a\rangle_{[0,2)}$ are equivalent, while neither of the two is equivalent to $\langle a\rangle_{[0,1)}\langle a\rangle_{[0,1)}$.

The main step is to construct a syntactic form of bisimulation, that relates not the languages denoted by timed regular expressions, but the timed regular expressions themselves. To do this, we need a few definitions. We say that two

timed regular expressions e_1 and e_2 are *equal up to ACI properties*, written $e_1 \stackrel{\text{ACI}}{=} e_2$, if e_1 and e_2 are syntactically equal, up to the associativity, commutativity, and idempotence of $+$. That is, e_1 and e_2 are equal up to ACI properties if the following three rewriting rules can be applied to subexpressions of e_1 to obtain e_2:

$$e + (f + g) = (e + f) + g$$
$$e + f = f + e$$
$$e + e = e.$$

Given a relation $\hat{R}$ between timed regular expressions, the induced relation $\hat{R}^{\text{ACI}}$ is defined by taking $e_1 \hat{R}^{\text{ACI}} e_2$ if and only if there exists e_1', e_2' such that $e_1 \stackrel{\text{ACI}}{=} e_1'$, $e_2 \stackrel{\text{ACI}}{=} e_2'$, and $e_1' \hat{R} e_2'$.

A *syntactic bisimulation with respect to* $T \subseteq \mathbb{R}_{\geq 0}$ between timed regular expressions e_1 and e_2 is a relation $\hat{R}$ on pairs of timed regular expressions such that

(1) $e_1 \hat{R} e_2$;
(2) if $e \hat{R} e'$, then $\hat{\epsilon}(e) = \hat{\epsilon}(e')$;
(3) if $e \hat{R} e'$, then for all $t \in T$ and all $a \in \Sigma$, $\hat{D}_{t,a}(e) \hat{R}^{\text{ACI}} \hat{D}_{t,a}(e')$.

A syntactic bisimulation resembles a bisimulation, except that it is defined over timed regular expressions, rather than over timed languages, and that the derivatives need only be taken with respect to times specified by a subset of $\mathbb{R}_{\geq 0}$. The next results show that any two equivalent timed regular expressions (of a specific kind) are related by a *finite* syntactic bisimulation, that is, a syntactic bisimulation $\hat{R}$ where the number of pairs in $\hat{R}$ is finite. This is surprising, because we can a priori take derivatives with respect to arbitrary positive real numbers, which seem to indicate that we need infinitely many tuples in $\hat{R}$, one per possible derivative. However, it turns out that we need only consider a finite (albeit perhaps large) number of derivatives when trying to establish the equivalence of any two particular timed regular expressions. Intuitively, for a certain kind of timed regular expression e, it suffices to look at derivatives with respects to a finite set of time increment.

Our theorem applies to timed regular expressions where every subexpression $\langle e \rangle_I$ is such that I is $[a, \infty)$ or $[a, b)$ for $a, b \in \mathbb{Q}_{\geq 0}$, that is, where every interval appearing in the expression is rational-bounded, left-closed, and right-open. Given two timed regular expressions e_1, e_2 of that form, we want to decide if $[\![e_1]\!] = [\![e_2]\!]$. We first show that for the purposes of establishing such an equality, it is sufficient to look at expressions with integer-bounded intervals. Intuitively, this is because we can "clear the denominator" on the bounds of all the intervals of e_1 and e_2 while preserving equivalence. Define the function

$Q(e)$ on timed regular expressions that computes a common denominator for all the interval bounds in e: let

$$\mathcal{I}(e) = \{I \mid \langle e' \rangle_I \text{ is a subexpression of } e \text{ for some } e'\},$$

and let

$$Q(e) = (\prod_{[p/q,\infty) \in \mathcal{I}(e)} q)(\prod_{[p_1/q_1,p_2/q_2) \in \mathcal{I}(e)} q_1 q_2).$$

Clearly, $Q(e)$ is a natural number. Given a natural number q, let $e \star q$ be the timed regular expression where every interval bound in e is multiplied by q:

$$\begin{aligned}
0 \star q &= 0 \\
1 \star q &= 1 \\
a \star q &= a \\
(e_1 e_2) \star q &= (e_1 \star q)(e_2 \star q) \\
(e_1 + e_2) \star q &= (e_1 \star q) + (e_2 \star q) \\
(e^*) \star q &= (e \star q)^* \\
(\langle e \rangle_{[a,\infty)}) \star q &= \langle e \star q \rangle_{[qa,\infty)} \\
(\langle e \rangle_{[a,b)}) \star q &= \langle e \star q \rangle_{[qa,qb)}
\end{aligned}$$

Proposition 6.1 *For e_1 and e_2 timed regular expressions subject to the above restrictions (every interval in e_1 and e_2 is rational-bounded, left-closed, and right-open), $[\![e_1]\!] = [\![e_2]\!]$ if and only if $[\![(e_1 \star Q(e_1)) \star Q(e_2)]\!] = [\![(e_2 \star Q(e_1)) \star Q(e_2)]\!]$.*

Note that all intervals in $(e_1 \star Q(e_1)) \star Q(e_2)$ and $(e_2 \star Q(e_1)) \star Q(e_2)$ are interger-bounded. Thus, it is sufficient to establish our result for timed regular expressions with integer bounds (and where every interval is left-closed and right-open, as before). This is the main result of this section. First, define the function $\Theta(e)$:

$$
\begin{aligned}
\Theta(0) &= 1 \\
\Theta(1) &= 1 \\
\Theta(a) &= 1 \\
\Theta(e_1 e_2) &= \max(\Theta(e_1), \Theta(e_2)) \\
\Theta(e_1 + e_2) &= \max(\Theta(e_1), \Theta(e_2)) \\
\Theta(e^*) &= \Theta(e) \\
\Theta(\langle e \rangle_{[a,\infty)}) &= \max(\Theta(e), a) \\
\Theta(\langle e \rangle_{[a,b)}) &= \max(\Theta(e), b)
\end{aligned}
$$

Theorem 6.2 *For all timed regular expressions e_1 and e_2 subject to the above restrictions (every interval in e_1 and e_2 is integer-bounded, left-closed, and right-open), $[\![e_1]\!] = [\![e_2]\!]$ if and only if there exists a finite syntactic bisimulation with respect to $T(e_1, e_2) = \{0, 1, 2, \ldots, \max(\Theta(e_1), \Theta(e_2))\}$ between e_1 and e_2.*

The "if" direction of the proof of Theorem 6.2 is the difficult one. From the finite syntactic bisimulation, we must show that we can always construct a full bisimulation relating the languages $[\![e_1]\!]$ and $[\![e_2]\!]$. To do this, we need to somehow "fill the holes" in the bisimulation, since the syntactic bisimulation is only defined with respect to a discrete set of derivatives (that is, with respect to the times in $T(e_1, e_2)$). If $\hat{R}$ is the given finite syntactic bisimulation, we can define R essentially as follows: $(L_1, \mathbf{v})R(L_2, \mathbf{v})$ if there exists $t' < 1$ and timed regular expressions e_1', e_2' obtainable by repeated applications of $\hat{D}_{t,a}$ for $t \in T(e_1, e_2)$ such that $D_{t'}(L_1) = [\![e_1']\!]$, $D_{t'}(L_2) = [\![e_2']\!]$ and $e_1' \hat{R} e_2'$. Thus, two timed languages are R-related if they correspond to timed regular expressions that are $\hat{R}$-related, except that they may have an additional small delay at the beginning of each timed word. The "only if" direction of the proof is simpler, and proceeds by constructing an appropriate syntactic bisimulation. The first step is to construct, for each timed regular expression e_1 and e_2, a finite-state machine with states timed regular expressions (up to ACI properties), and where the transitions correspond to taking derivatives $\hat{D}_{t,a}$ for $t \in T(e_1, e_2)$ and $a \in \Sigma$. This finite-state machine can be defined by induction on the structure of e. Roughly speaking, the machine captures the timed regular expressions obtainable from e by taking one or more derivatives. Given such finite-state machines M_1 and M_2 for e_1 and e_2, a finite syntactic bisimulation $\hat{R}$ can be constructed as follows. Initialize $\hat{R}$ to contain the pair (e_1, e_2), and iterate the following process: for every (e, e') in $\hat{R}$, add the pairs obtained by taking all transitions from e in M_1 and from e' in M_2. Perform this iteration until no new pairs are added to $\hat{R}$. This must terminate, because there are finitely many

pairs of states (e, e') with e in M_1 and e' in M_2. It is straightforward to check that $\hat{R}$ is a syntactic bisimulation, under the assumption that $[\![e_1]\!] = [\![e_2]\!]$.

This procedure can be easily turned into a procedure for deciding if two timed regular expressions are equivalent: construct $\hat{R}$, and verify that at all pairs (e, e') in $\hat{R}$, $\hat{\epsilon}(e) = \hat{\epsilon}(e')$. The two timed regular expressions are equivalent if and only if this verification succeeds.

There are two obvious open questions that remain to be resolved. First, can Theorem 6.2 be extended to real-bounded intervals? Preliminary investigations seem to indicate that it is the case, but the details remain to be worked out. Second, can Theorem 6.2 be extended to timed regular expressions with other kinds of integer-bounded intervals? A variation of the proof of Theorem 6.2 can be used to get a version of the theorem for expressions e_1 and e_2 with interval-bounded, left-open, and right-closed intervals. Whether this can be extended to expressions with a mix of both types of intervals, or with left-open and right-open intervals is not clear.

7 Conclusion

We have initiated, in this paper, a study of the coinductive properties of deterministic timed automata, paralleling the work of Rutten [14] on classical automata. This lets us derive a coinduction proof principle for timed languages, which yields an algorithm for deciding whether two timed regular expressions (of a specific kind) are equivalent. The latter is surprising, since a priori, the bisimulation that one is required to exhibit to apply the coinduction proof principle appears to be infinite. Our work fits in the program set out by Trakhtenbrot [17] to lift the results of classical automata theory and languages to real-time models.

We have restricted ourselves to studying the uncontroversial subclass of timed regular expressions consisting of regular expressions extended with a $\langle e \rangle_I$ operator that restricts the duration of the timed words corresponding to e. It remains to be seen whether the approach applies to any of the extended form of timed regular expressions described in Section 3 that can capture the full class of languages recognized by the timed automata of Alur and Dill [1].

There been a fair amount of work on studying bisimulation of timed processes in the context of timed process algebras (see [18,12], for instance.) It would be interesting to relate that work to the one in this paper. One difference is that timed processes are typically not taken to be distributive (that is, $\cdot$ does not distribute over $+$), while timed regular expressions are. The work of Corradini et al. [8] may be relevant for understanding this difference.

On a more general note, we now have at least three examples of results

relating a form of regular expression with a form of languages and deterministic automata, yielding an algorithm to establish the equivalence of expressions: classical regular expressions [14], Kleene algebra with tests expressions [7], and the current result on timed regular expressions. We leave as an open question whether there is a general result relating regular expressions, languages, and automata, of which the cited results can be seen as instances.

Acknowledgments

The ideas in Section 6 owe much to collaborative work with Hubie Chen. This work was supported in part by NSF under grant CTC-0208535, by ONR under grand N00014-02-1-0455, by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the ONR under grant N00014-01-1-0795, and by AFOSR under grant F49620-02-1-0101.

References

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

[2] Asarin, E., P. Caspi and O. Maler, *A Kleene theorem for timed automata*, in: *Proc. 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)* (1997), pp. 160–171.

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

[4] Asarin, E. and C. Dima, *Balanced timed regular expressions*, in: *Proc. 3rd International Workshop on Models for Time-Critical Systems (MTCS'02)*, Electronic Notes in Theoretical Computer Science **68.5** (2002).

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

[6] Brzozowski, J. A., *Derivatives of regular expressions*, Journal of the ACM **11** (1964), pp. 481–494.

[7] Chen, H. and R. Pucella, *A coalgebraic approach to Kleene algebra with tests*, in: *Proc. 6th International Workshop on Coalgebraic Methods in Computer Science*, Electronic Notes in Theoretical Computer Science **82.1** (2003).

[8] Corradini, F., R. de Nicola and A. Labella, *Models of nondeterministic regular expressions*, Journal of Computer and System Sciences **59** (1999), pp. 412–449.

[9] Dima, C., *Regular expressions with timed dominoes*, in: *Proc. 4th International Conference on Discrete Mathematics and Theoretical Computer Science (DMTCS'03)*, Lecture Notes in Computer Science **2731** (2003), pp. 141–154.

[10] Herrmann, P., *Renaming is necessary in timed regular expressions*, in: *Proc. Conference on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science **1738** (1999), pp. 47–59.

[11] Hopcroft, J. E. and J. D. Ullman, "Formal languages and their relation to automata," Addison Wesley, 1969.

[12] Kick, M., *Bialgebraic modelling of timed processes*, in: *Proc. 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, Lecture Notes in Computer Science **2380**, 2002, pp. 525–536.

[13] Kozen, D., *A completeness theorem for Kleene algebras and the algebra of regular events*, Information and Computation **110** (1994), pp. 366–390.

[14] Rutten, J. J. M. M., *Automata and coinduction (an exercise in coalgebra)*, in: *Proc. 9th International Conference on Concurrency Theory (CONCUR'98)*, Lecture Notes in Computer Science **1466**, 1998, pp. 193–217.

[15] Rutten, J. J. M. M., *Universal coalgebra: a theory of systems*, Theoretical Computer Science **249** (2000), pp. 3–80.

[16] Salomaa, A., *Two complete axiom systems for the algebra of regular events*, Journal of the ACM **13** (1966), pp. 158–169.

[17] Trakhtenbrot, B., *Origins and metamorphoses of the trinity: Logic, nets, automata*, in: *Proc. 10th Annual IEEE Symposium on Logic in Computer Science (LICS'95)* (1995), pp. 506–507.

[18] Wang, Y., *Real-time behaviour of asynchronous agents*, in: *Proc. 2nd International Conference on Concurrency Theory (CONCUR'90)*, Lecture Notes in Computer Science **458** (1990), pp. 502–520.

[19] Wilke, T., *Specifying state sequences in powerful decidable logics and timed automata*, in: *Proc. 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science **863** (1994), pp. 694–715.