Kleene Theorems for Event-Clock Automata

Cătălin Dima, *

Bucharest University, Department of Foundations of Computer Science

Abstract. We define some class of regular expressions equivalent to event-clock automata. It is shown that regular expressions cannot be given a compositional semantics in terms of timed state sequences. We introduce a modified version of timed state sequences supporting a *partial* operation of concatenation on which we may build the semantics of regular expressions. A *forgetting* map then induces a semantics in terms of the classic version of timed state sequences. We also define several types of languages of automata in terms of classic or modified timed state sequences. Two Kleene theorems, one for each type of timed state sequences, relating expressions and event-clock automata are proved.

1 Introduction

Automata models of real-time systems have proved instrumental in the verification of real-time systems [AD94,AFH94]. They have been studied with the hope that the classical decidability results and the relationship to temporal logics can be adapted in the real-time framework, hence giving powerful decidable real-time logics. However the program has not reached its scope, since there is no fully satisfactory class of automata fitting the above requirements. Timed automata of [AD94] are too powerful, having a decidable emptiness problem but an undecidable complementation problem. Real-time automata [Di2], in spite of their decidability of the complementation problem, have too little expressive power, being unable to capture behaviors of distributed real-time systems [Di3]. Another proposal are event-clock automata [AFH94] (ECA for short), which, in their introductory paper, were shown to have a decidable complementation problem as they are determinizable. These automata have also sufficient expressive power and have served as a guide for defining some real-time logic [RS97,HRS98].

Another subject for research was the relationship of automata to some regular expressions. As yet just timed automata [ACM97] and real-time automata [Di2] have been the subject of this question, but only real-time automata have proved satisfactory since as (languages accepted by) timed automata have a weak Kleene theorem in the sense that it involves renaming, and it is still an open question if regular expressions in [ACM97] are equally expressive to timed automata.

Here we investigate the class of languages accepted by ECA for their relationship to some regular expressions. The expressions in study have the same form as the labels of the transitions on ECA, i.e. are composed of a letter or the empty word ϵ and a constraint on the history and prophecy clocks. We call

* This research was done during the author's visit to TIFR, Bombay, supported by an extension of a UNU/IIST fellowship. Mailing address: str. Academiei 14, Bucharest, Romania, R0-70109, e-mail: `cdima@funinf.math.unibuc.ro`

concatenation of these expressions *chop*. It turns out that the constraints on the left-hand side of chop may refer to points after the chopping point. We are lead to consider a modified version of timed state sequences of [AD94], in which we add information about some interval to which attention is restricted (we call them TSS *with limited observation*). This allows us to define the chop of two TSS with limited observation only iff they have the same underlying TSS, hence chop being a partial operation. This is then lifted to a total operation on sets of TSS with limited observation. We also show that, if we assume the idea that constraints on one part of chop may refer to points on the other part of chop, we cannot define semantics of the regular expressions in terms of a concatenation operation on TSS.

We also study here ECA with ϵ-transitions, not yet defined in the litterature. Why ϵ-transitions, if ECA are not closed under renaming and hiding [AFH94]? The reason lies in their relationship to the logic proposed in [RS97,HRS98]. Take e.g. the formula $\Diamond \triangleright_{=1} p$ (which is to be read as "there is some time point t such that the first time point $t' > t$ at which p starts to hold must satisfy $t' - t = 1$"). The formula might be associated to the state-clock automaton in figure 1 (see [RS97,HRS98] for details).

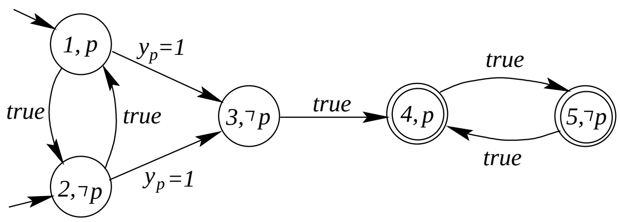

Fig. 1

Intuitively the automaton may be in states 1 or 4 at some time point τ only if the value of variable p is true at τ, while during the stay in states 2, 3 or 5 the value of p must be false. Some transition may be taken at time point τ iff the values of the variable p before and after τ fit the respective labels of the states and the constraint on the transition is true at τ. The constraint $y_p = 1$ is true at some time point τ iff the next time point τ' at which p becomes *true* satisfies $\tau' - \tau = 1$ (this implies that p is false at τ). The automaton is built such that it stays in states 1 or 2 before the time point t; at moment t it goes to 3 and at time point t' it goes to 4; after this it stays either in 4 or in 5. Hence it is equivalent to the above formula. Note that transitions 1 to 2, 2 to 1 etc. involve also a change in the value of p; but there is also the transition 2 to 3 which is identified only by the moment at which the constraint becomes true. We argue that, when transforming this automaton into an ECA, the transition from 2 to 3 has to be an ϵ-transition by the intuition that *an action (i.e. letter) occurs only when some variable changes its value.*

Note then that in the presence of ϵ-transitions the algorithm in [AFH94] does not give a deterministic ECA as the ϵ-transitions have to be removed first - the same holds in the classical setting [HU]. In fact the notion of deterministic automaton of [AFH94] has to be redefined. The problem of the elimination of ϵ-transitions is solved in [Di4].

The paper runs as follows: we introduce in the next section extended TSS (ETSS for short) which allow us to define the language of an ECA with ϵ-transitions and ETSS with limited observation (LETSS for short) with the partial concatenation ("chop") on them which allows defining the semantics of regular expressions. In the third section we define ECA with ϵ-transitions and ϵ-regular expressions. We associate two types of semantics to regular expressions, one in terms of LETSS and one in terms of TSS. Only the first one is compositional, the other one is not and comes via some "forgetting" map. Also we associate to each ECA four languages: the first language, in terms of ETSS, is the natural one, as it extends the definition of [AFH94] to ECA with ϵ-transitions; the second language, in terms of TSS comes via the forgetting map. But, since we cannot prove the Kleene theorems unless we define chop on automata too we also need some language in terms of LETSS, which is the third type. The fourth type language is again in terms of TSS and comes from the third via some forgetting map too. An important property showed here is that the second and the fourth types of languages, though being not the same for a given ECA, have equal expressive power. The fourth section contains the proof of the Kleene theorem relating the first type semantics of some ϵ-regular expression and the third type language of some ECA. As a corollary we obtain the (desired) Kleene theorem saying that ϵ-regular expression and ECA have the same expressive power w.r.t. their TSS semantics. We end with some comments and topics for further study.

2 Preliminaries

We fix a finite set of letters Σ. For a word $w \in \Sigma^*$ we denote by $|w|$ its length. $[n]$ is the set of natural numbers $\{1, \dots, n\}$. Notations $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$ are for the sets of nonnegative, resp. positive reals. Similarily $\mathbb{Q}_{\geq 0}$ stands for the set of nonnegative rationals.

Definition 1. *An* ***extended timed state sequence*** *(*ETSS *for short) is a tuple (w, τ, I) consisting of a word $w \in \Sigma^*$, a strictly increasing function $\tau : [m] \longrightarrow \mathbb{R}_{\geq 0}$ with $m \geq |w|$, called the* ***sequence of time points of the actions*** *and a subset I of $[m]$ having $card(I) = |w|$.*

We denote $dom(\tau)$ the domain $[m]$ of the sequence of time points τ and by $range(\tau)$ the range of τ.

An ETSS intuitively models the behavior of a real-time system in the following sense: the letters in Σ are the *observable actions*, the empty word ϵ represents an *inobservable action* and τ holds the time points at which actions occur. Moreover observable actions occur only at $\tau(i)$ with $i \in I$. Hence, if we consider that I consists of $j_1 < j_2 < \dots < j_{|w|}$ we can define, for each $i \in dom(\tau)$, the i-th symbol of the ETSS as being ϵ (the empty word) iff $i \notin I$ or the k-th letter in w iff $i = j_k \in I$ and denote this by $w[i]$. Then $w = w[1]w[2] \dots w[card(I)]$.

A natural equivalence arises on ETSS: two ETSS (w_i, τ_i, I_i), $i \in [2]$ are equivalent, denoted $(w_1, \tau_1, I_1) \simeq (w_2, \tau_2, I_2)$, iff $w_1 = w_2$ and the time points of observable actions are the same, i.e. if we list each I_i in increasing order as $I_i = \{j_1^i, \ldots, j_{|w^1|}^i\}$ then $\tau_1(j_k^1) = \tau_2(j_k^2)$ for all $k \in [|w_1|]$.

A representative in each equivalence class can be defined as follows: for an ETSS $\alpha = (w, \tau, I)$, where I is listed in increasing order as $I = \{j_1, \ldots, j_{|w|}\}$), we define a function $\theta : [|w|] \longrightarrow \mathbb{R}_{>0}$ with $\theta(k) = \tau(j_k)$ for all $k \in [|w|]$. If we pair the word w with θ we actually get the following definition:

Definition 2. *A **timed state sequence** [AD94] (*TSS *for short) is a pair (w, θ) consisting of a word $w \in \Sigma^*$ and a strictly increasing function $\theta : [|w|] \longrightarrow \mathbb{R}_{\geq 0}$.*

Hence in a TSS we know only the time points of the observable actions. Clearly they are special kind of ETSS, in which $I = dom(\tau)$. Simple verification shows that the map $[\cdot]$ from the set of ETSS to the set of TSS defined as $[(w, \tau, I)] = (w, \theta)$ where θ is defined above maps two equivalent ETSS to the same TSS. Thus TSS can be thought of as representatives for each equivalence class of ETSS.

On ETSS we can also define concatenation as an extension of the concatenation on words: given two ETSS (w_1, τ_1, I_1) and (w_2, τ_2, I_2) with $dom(\tau_i) = [m_i]$ $(i \in [2])$ define $(w_1, \tau_1, I_1) \cdot (w_2, \tau_2, I_2) = (w_1 w_2, \tau, I)$ where $dom(\tau) = [m_1 + m_2]$ and

$$\tau(i) = \begin{cases} \tau_1(i) & \text{iff } i \in [m_1] \\ \tau_1(m_1) + \tau_2(i - m_1) & \text{iff } m_1 + 1 \leq i \leq m_1 + m_2 \end{cases}$$

Concatenation can then be extended to sets of ETSS as usual. We denote by $\cdot$ this extension. Concatenation can be defined on TSS too, as TSS are special kind of ETSS.

A third basic notion is the following:

Definition 3. *An ETSS **with limited observation** (LETSS for short) is a tuple (w, τ, I, t_1, t_2) where (w, τ, I) is an ETSS and $0 \leq t_1 \leq t_2 < \infty$*

The intuition is that observation of the system is restricted to the interval $[t_1, t_2]$. LETSS are essential in giving semantics to regular expressions.

The *underlying* ETSS of a LETSS is defined by $sub((w, \tau, I, t_1, t_2)) = (w, \tau, I)$. The equivalence relation on ETSS can be straightforwardly extended to LETSS by putting two LETSS equivalent when their underlying ETSS are. Then we can define a map $[[\cdot]]$ as the composition of sub with $[\cdot]$. This map also makes TSS as representatives of equivalent LETSS.

We can also define concatenation of LETSS, but in a rather different manner than on ETSS: it is a *partial* operation, defined when both operands have the same underlying ETSS and when the endpoint of the interval of observation of the first equals the initial point of the interval of observation of the second:

$$(w, \tau, I, t_1, t_2); (w', \tau', I', u_1, u_2) = \begin{cases} (w, \tau, I, t_1, u_2) & \text{iff } w = w', \tau = \tau', I = I', t_2 = u_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

This operation is then extended to a *total* operation on sets of LETSS:

$$T_1;T_2 = \{(w,\tau,I,t_1,t_3) \mid \exists t_2 \in (t_1,t_3) \text{ such that } (w,\tau,I,t_1,t_2) \in T_1 \text{ and } (w,\tau,I,t_2,t_3) \in T_2\}$$

Note that $\{(w,\tau,I,t_1,t_2)\};\{(w,\tau,I,t_2,t_3)\} = \{(w,\tau,I,t_1,t_3)\}$ while $\{(w,\tau,I)\} \cdot \{(w,\tau,I)\} = \{(ww,\tau',I')\}$ for some τ' and I'. Hence

Remark 1. Given two sets of LETSS T_1 and T_2, $sub(T_1;T_2)$ does not equal $sub(T_1) \cdot sub(T_2)$ in general.

In the sequel we remind the use of clocks in association with an ETSS:

We associate with each letter a of Σ two symbols, a **history clock** x_a and a **prophecy clock** y_a. Denote then $H(\Sigma) = \{x_a \mid a \in \Sigma\}$ and $P(\Sigma) = \{y_a \mid a \in \Sigma\}$. the respective copies of Σ that arise this way[1]. The use of the clocks is connected to each ETSS as follows: at each time point $t \in \mathbb{R}_{\geq 0}$ we can define the value of the clock x_a as the time elapsed since the last occurence of letter a in the ETSS; if no such occurences, put $x_a = \bot$. Similarily, y_a holds the time remaining till the next occurence of a and is $\bot$ if there is no such occurence. Hence some ETSS (w,τ,I) defines a **trajectory of clocks** which is a function $Ck = Ck_{((w,\tau,I),t_1)} : (H(\Sigma) \cup P(\Sigma)) \times \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0} \cup \{\bot\}$ defined as:

$$Ck(x_a,t) = \begin{cases} t-\tau(i) & \text{iff, denoting } A = \{k \in I \mid w[k]=a \text{ and } \tau(k)<t\} \text{ we have} \\ & \quad A \neq \emptyset \text{ and } i = \max A \\ \bot & \text{otherwise} \end{cases}$$

$$Ck(y_a,t) = \begin{cases} \tau(i)-t & \text{iff, denoting } B = \{k \in I \mid w[k]=a \text{ and } \tau(k)>t\} \text{ we have} \\ & \quad B \neq \emptyset \text{ and } i = \min B \\ \bot & \text{otherwise} \end{cases}$$

A **clock constraint** is a boolean combination of atomic formulas of the type $x_a < c$, $x_a \leq c$, $x_a = \bot$, and $y_a < c$, $y_a \leq c$ or $y_a = \bot$ where c is some positive rational $c \in \mathbb{Q}_{\geq 0}$. The set of clock constraints over Σ is denoted $Constr(\Sigma)$. For each initialized ETSS $((w,\tau,I),t_1)$ the trajectory of clocks Ck defines at each time point $t \in \mathbb{R}_{\geq 0}$ an interpretation of the clock constraints. We use the notation $(w,\tau,I,t_1,t) \models C$ to encode that the interpretation $Ck(\cdot,t)$, maps the constraint C to true.

For each clock $K \in H(\Sigma) \cup P(\Sigma)$ define the **elementary clock constraints** as all the formulas $K = \bot$, $c < K$, $c \leq K$, $c_1 < K \wedge K < c_2$, $c_1 \leq K \wedge K < c_2$, $c_1 < K \wedge K \leq c_2$ or $c_1 \leq K \wedge K \leq c_2$, for some $c, c_1, c_2 \in \mathbb{Q}_{\geq 0}$. A **simple constraint** is a formula $\bigwedge_{a \in \Sigma}(C(x_a) \wedge C(y_a))$ where $C(x_a)$ and $C(y_a)$ are elementary constraints. We denote $SC(\Sigma)$ the set of simple constraints.

It is clear that any clock constraint can be brought to a disjunction of simple clock constraints (a normal form without negation). E.g. $\neg(x_a < 2) \equiv (x_a \geq 2) \vee (x_a = \bot)$. This property will be used in simplifying event-clock automata.

[1] Note that we do not use clocks x_ϵ or y_ϵ. These clocks may be regarded as event-clock versions for the automata clocks in [HRS98].

3 Event-clock automata and regular expressions

In this his section we introduce event-clock automata with ϵ-transitions and regular expressions which match these automata. The introduction of ϵ-transitions in event-clock automata justifies the introduction of extended time state sequences.

Definition 4. *An **event-clock automaton** (ECA for short) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where Q is the set of* states*, $Q_0, F \subseteq Q$ are the sets of* initial *resp.* final states *and δ is the* transition relation $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Constr(\Sigma) \times Q$ *with $|\delta| < \infty$.*

A **run** of length n is a sequence of transitions $((q_{i-1}, x_i, C_i, q_i))_{i \in [n]}$ ($x_i \in \Sigma \cup \{\epsilon\}$) such that $q_0 \in Q_0$ and $q_n \in F$. A zero-length run consists of some initial and final state. A **partial run** is a sequence of transitions like above but which may start or end in any location.

We will give four types of languages to ECA. The first one is in terms of ETSS: an ETSS (w, τ, I) with $dom(\tau) = [n]$ is accepted by $\mathcal{A}$ iff there is a run (of the same length) $((q_{i-1}, x_i, C_i, q_i))_{i \in [n]}$ such that the word defined by the run is w and at the $i - th$ action in the ETSS the respective constraint is true. Formally, for all $i \in [n]$ we have that $w[i] = x_i$ and $(w, \tau, I, \tau(i)) \models C_i$. We call some run *admissible* iff it is associated to some ETSS in the above sense. We denote $L_e(\mathcal{A})$ the set of ETSS accepted by $\mathcal{A}$ and call it the **extended language** of $\mathcal{A}$.

The second semantics comes via the map $[\cdot]$: the **abstract language** of $\mathcal{A}$, is the set of TSS which are the representatives of some ETSS accepted by $\mathcal{A}$: $L_a(\mathcal{A}) = [L_e(\mathcal{A})] = \{[(w, \tau, I)] \mid (w, \tau, I) \in L_e(\mathcal{A})\}$

The third semantics is in terms of LETSS: a LETSS (w, τ, I, t_1, t_2) with $dom(\tau) = [n]$ is accepted by $\mathcal{A}$ iff there is a run $((q_{i-1}, x_i, C_i, q_i))_{i \in [m]}$ such that m is the number of actions that occur within the interval $[t_1, t_2]$ and at the i-th action occuring in $[t_1, t_2]$ constraint C_i is satisfied. Formally, $m = card(\{i \in [n] \mid \tau(i) \in [t_1, t_2]\}$ and, denoting $j = \min\{i \in [m] \mid \tau(i) \in [t_1, t_2]\}$ we have that $(w, \tau, I, \tau(j + i - 1)) \models C_i$ for all $i \in [n]$. We denote $L_l(\mathcal{A})$ the set of LETSS accepted by $\mathcal{A}$ and call it the **limited observation language**.

The last semantics, the **language of abstract limited observation** is $L_{al}(\mathcal{A}) = [[L_l(\mathcal{A})]] = \{[[(w, \tau, I, t_1, t_2)]] \mid (w, \tau, I, t_1, t_2) \in L_l(\mathcal{A})\}$.

Note that $L_a(\mathcal{A}) \neq L_{al}(\mathcal{A})$ in general, a counterexample being the ECA $\mathcal{A} = (\{q, r\}, \{a\}, \{(q, a, true, r)\}, \{q\}, \{r\})$ for which L_{al} is the set of all TSS having at least one letter, while $L_a(\mathcal{A})$ is a singleton. However

Proposition 1. *The class of sets of TSS which are the abstract language of some ECA equals the class of sets of TSS which are the language of abstract limited observation of some ECA.*

The proof of this is based on the following normal form of automata:

Proposition 2. *For each ECA $\mathcal{A}$ there exist an ECA $\mathcal{B}$ such that with $L_e(\mathcal{A}) = L_e(\mathcal{B})$ and such that all transitions in $\mathcal{B}$ have simple constraints (call this automaton **simple ECA**).*

Proof. We rely on the decomposition of each constraint into a disjunction of simple constraints mentioned at the end of the previous section. We decompose first each constraint C on some transition into n simple constraints and then split that transition into n transitions with the same source and destination, each labeled with one of the simple constraints C is decomposed into. ☐

The proof of Proposition 1 can be done as follows: First observe that in a simple ECA an initial transition (i.e. starting in an initial state) which is labeled with a simple constraint C is in an admissible run iff C constrains all the history clocks to $\perp$ and similarily for final transitions (i.e. leading to some final state) and prophecy clocks. Note that this does not hold for runs associated to LETSS: e.g. the ECA $(\{q,r\},\{a\},\{(q,\epsilon,x_a=2,r)\},\{q\},\{r\})$ has no admissible run but has some run associated to some LETSS (w,τ,I,u_1,u_2) in which $range(\tau)\cap[t_1,t_2]=\{\tau(i)\}$, $w[i]=\epsilon$ and $(w,\tau,I,\tau(i))\models x_a=2$.

Hence, for the left-to-right passage transform the given ECA $\mathcal{A}$ (assumed simple) such that initial states have no incoming transitions and final states have no outgoing transitions (by the state-splitting technique). Then remove all initial transitions whose (simple) constraints do not check some history clock x_a to $\perp$; similarily remove all final transitions on which some prophecy clock y_a is not constrained to $\perp$. Note that the resulting ECA has the property that it accepts a LETSS (w,τ,I,t_1,t_2) iff the interval $[t_1,t_2]$ contains the domain of τ. Finally, for all the remaining initial transitions remove from their constraints all conjuncts that refer to history clocks (which may be only $K=\perp$), keeping the conjuncts that refer to prophecy clocks; similarily, for all the remaining final transitions remove from their constraints the conjuncts that refer to prophecy clocks. We get an ECA $\mathcal{B}$ with $L_{al}(\mathcal{B})=L_a(\mathcal{A})$.

What for the reverse, starting with some ECA $\mathcal{A}$ (not necessarily simple) transform it first such that initial states do not have incoming transitions and final states do not have outgoing transitions. Then, at all initial and final states, add loops labeled with all letters in Σ and constraint *true*. The abstract language of the resulting automaton will be equal to the language of abstract limited observation of $\mathcal{A}$. ☐

Definition 5. *An ECA is **ϵ-free** iff there is no tuple in the transition relation labeled with the empty word. An ϵ-free ECA is **deterministic** iff it has a single initial state and for each two distinct transitions having the same source and labeled with the same symbol the conjunction of their constraints is a nonsatisfiable formula.*

An ϵ-free ECA can be brought to a deterministic one by the algorithm in [AFH94], which is an adaptation of the subset construction. As we have noted in the introduction this construction does not work for ECA that have ϵ-transitions: this is similarily to the situation in finite automata theory. Bringing an ECA to an ϵ-free one is the subject of [Di4].

Definition 6. *The set $\epsilon RE(\Sigma)$ of **ϵ-regular expressions** over Σ is defined by the following grammar where $a\in\Sigma$ and $C\in Constr(\Sigma)$:*

$$E ::= \emptyset \mid () \mid (\epsilon,C) \mid (a,C) \mid E+E \mid E;E \mid E^*$$

The rules without ϵ give the set of ***regular expressions****.*

We call *atomic* expressions of the type (x, C) with $x \in \Sigma \cup \{\epsilon\}$. Also $()$ is called the *empty expression*.

We define two types of semantics of ϵ-regular expressions, namely one in terms of LETSS (the *concrete* semantics) and another one in terms of TSS (the *abstract* semantics). The *concrete semantics* of an ϵ-regular expression E is denoted $|E|$ and for atoms is:

- $|()|$ containts all the LETSS (w, τ, I, t_1, t_2) such that the interval $[t_1, t_2]$ does not contain any action (i.e. $range(\tau) \cap [t_1, t_2] = \emptyset$).
- $|(\epsilon, C)|$ contains some LETSS (w, τ, I, t_1, t_2) iff there is a unique time point $\tau(i) \in [t_1, t_2]$ at which the inobservable action occurs, C holds there and no other action occurs in $[t_1, t_2]$ (i.e. $range(\tau) \cap [t_1, t_2] = \{\tau(i)\}$, $(w, \tau, I, \tau(i)) \models C$ and $w[i] = \epsilon$).
- $|(a, C)|$ contains some LETSS (w, τ, I, t_1, t_2) iff there is a unique time point $\tau(i) \in [t_1, t_2]$ at which the observable action a occurs, C holds there and no other action occurs in the interval $[t_1, t_2]$ (i.e. $range(\tau) \cap [t_1, t_2] = \{\tau(i)\}$, $(w, \tau, I, \tau(i)) \models C$ and $w[i] = a$).

The semantics is then easily extended to ϵ-regular expressions by structural induction:

$$|E_1; E_2| = |E_1|; |E_2| \qquad |E_1 + E_2| = |E_1| \cup |E_2| \qquad |\emptyset| = \emptyset$$
$$|E^*| = \bigcup_{n \in \mathbb{N}} |E^n| \text{ where } E^0 = () \text{ and } E^{n+1} = E^n; E$$

The *abstract* semantics, denoted $\|E\|$ comes via the map $[[\cdot]]$ defined in the previous section: we put $\|E\| = [[|E|]]$.

Hence there are two notions of equivalence of ϵ-regular expressions: a concrete one and an abstract one. Clearly two ϵ-regular expressions which are concrete equivalent are abstract equivalent too. The expressions $(\epsilon, (x_a = 2) \wedge (y_a = 4))$ and $(\epsilon, (x_a = 4) \wedge (y_a = 2))$ show the reverse does not hold.

Proposition 3. *Each ϵ-regular expression is concrete equivalent to an ϵ-regular expression in which all constraints are simple.*

We conclude this section by pointing out that the semantics of chop on ϵ-regular expressions cannot rely on concatenation of ETSS as defined above, hence ETSS cannot serve for defining some "natural" semantics of ϵ-regular expression. The exact meaning of *natural* has to be studied further, but we rely on some properties concatenation should have based on our intuition that constraints on the lhs of ";" may refer to time points after the chopping point. One such property is that the semantics of $(a, y_b = \perp); (b, C)$ should be the empty set regardless of the constraint C, but the semantics of (b, C) should be the empty set iff C would be some unsatisfiable constraint. Hence the operation that gives the semantics of chop should be able to capture the situation when applying it to nonempty sets gives the empty set, situation which is not captured by any operation which is defined at the level of elements (like "$\cdot$" on ETSS).

4 The two Kleene theorems

Theorem 1 (Concrete Kleene theorem). *The class of sets of LETSS which are the limited observation semantics of some ECA equals the class of LETSS which are the concrete semantics of some ϵ-regular expression.*

Proof. For the right-to-left inclusion we proceed by structural induction. Note first that, for each atomic ϵ-regular expression (x, C) $(x \in \Sigma \cup \{\epsilon\})$ the ECA $\mathcal{A}_{x,C} = (\{q, r\}, \Sigma, \{(q, x, C, r)\}, \{q\}, \{r\})$ has $L_l(\mathcal{A}_{x,C}) = |(x, C)|$. Also for the empty expression () we may consider a single state ECA with no transition: $(\{q\}, \Sigma, \emptyset, \{q\}, \{q\})$. Then we adapt the usual constructions [HU] for union, concatenation and star:

Fix two ECA $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, Q_0^i, F_i)$, $i \in [2]$ with disjoint sets of states $(Q_1 \cap Q_2 = \emptyset)$. Then the ECA $\mathcal{A}_\cup = (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2, Q_0^1 \cup Q_0^2, F_1 \cup F_2)$ has $L_l(\mathcal{A}_\cup) = L_l(\mathcal{A}_1) \cup L_l(\mathcal{A}_2)$.

What for chop, we note that drawing ϵ-transitions from final states of $\mathcal{A}_1$ to initial states of $\mathcal{A}_2$ does not work here since LETSS accepted by the resulting automaton would have an extra time point in τ, corresponding to the ϵ-transition. Hence we first transform both automata such that initial states do no have incoming transitions and final states do not have outgoing transitions. Then build the automaton $\mathcal{A}_; = (Q, \Sigma, \delta, Q_0, F)$ where

- Q consists of nonfinal states in $\mathcal{A}_1$, noninitial states in $\mathcal{A}_2$ and pairs $(q, r) \in F_1 \times Q_0^2$;
- Q_0 consists of the initial and nonfinal states of $\mathcal{A}_1$ and, for each initial and final state s in $\mathcal{A}_1$, all pairs (s, r) with r initial in $\mathcal{A}_2$;
- similarily, F consists of final and noninitial locations of $\mathcal{A}_2$ and, for each final and initial state s in $\mathcal{A}_2$, all pairs (r, s) with r final in $\mathcal{A}_1$;
- δ consists of all transitions from δ_1 that do not involve some final state and all transitions from δ_2 not involving some initial state; moreover, each transition in $\mathcal{A}_1$ leading to some final state r gives rise to transitions leading to all pair states containing r and each transition in $\mathcal{A}_2$ starting in some initial state r gives rise to transitions starting in all pair states containing r. Formally:

$$\begin{aligned}\delta = \{&(q, x, C, r) \mid (q, x, C, r) \in \delta_1 \text{ and } r \notin F_1\} \cup \\ \{&(q, x, C, r) \mid (q, x, C, r) \in \delta_1 \text{ and } q \notin Q_0^2\} \cup \\ \{&(q, x, C, (r, s)) \mid (q, x, C, r) \in \delta_1, r \in F_1 \text{ and } s \in Q_0^2\} \cup \\ \{&((q, r), x, C, s) \mid (r, x, C, s) \in \delta_2, r \in Q_0^2 \text{ and } q \in F_1\}\end{aligned}$$

The resulting automaton has $L_l(\mathcal{A}_;) = L_l(\mathcal{A}_1); L_l(\mathcal{A}_2)$.

The star construction is similar to the above, in the sense that each pair consisting of a final state and an initial one gives a new state. Hence we start, say, with $\mathcal{A}_1$ above, which is assumed to satisfy the requirement that initial states do not have incoming transitions and final states do not have outgoing transitions. Then construct $\mathcal{A}_* = (Q_*, \Sigma, \delta_*, Q_0^1, F_*)$ where Q_0^1 is taken from $\mathcal{A}_1$ and

$$Q_* = (Q_1 \setminus F_1) \cup (F_1 \times Q_0^1) \qquad F_* = F_1 \times Q_0^1$$

$$\begin{aligned}\delta = \{(q,x,C,r) \mid (q,x,C,r) \in \delta_1 \text{ and } r \notin F_1\} \cup \\ \{(q,x,C,(r,s)) \mid (q,x,C,r) \in \delta_1, r \in F_1 \text{ and } s \in Q_0^1\} \cup \\ \{((q,r),x,C,s) \mid (r,x,C,s) \in \delta_1, r \in Q_0^1 \text{ and } q \in F_1\}\end{aligned}$$

The resulting automaton has $L_l(\mathcal{A}_*) = (L_l(\mathcal{A}))^*$.

For the left-to-right inclusion, fix some ECA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ with $Q = \{q_1, q_2, \ldots, q_n\}$ and denote $\mathcal{A}_{ij} = (Q, \Sigma, \delta, \{q_i\}, \{q_j\})$. Build as in [HU] the sequence of ϵ-regular expressions $(E_{ij}^k)_{1 \le i,j,k \le n}$ whose concrete semantics equals the set of LETSS which have a run in $\mathcal{A}_{ij}$ whose intermediate states (i.e. excepting q_i and q_j) have indices less than k:

$$E_{ij}^0 = \begin{cases} \sum\limits_{(q_i,x,C,q_j)\in\delta} (x,C) & \text{iff } i \neq j \\ \sum\limits_{(q_i,x,C,q_j)\in\delta} (x,C) + () & \text{iff } i = j \end{cases}$$

$$E_{ij}^k = E_{ij}^{k-1} + E_{ik}^{k-1}; (E_{kk}^{k-1})^*; E_{kj}^{k-1}$$

Then $|\sum\limits_{q_i \in Q_0} \sum\limits_{q_j \in F} E_{ij}^n| = L_l(\mathcal{A})$ □

Note that the proof of this theorem may also run as follows: we can define *symbolic semantics* of ϵ-regular expressions as words over the set of symbols $\Omega = \Sigma \times Constr(\Sigma)$ that are the classical semantics of the ϵ-regular expression, e.g. the symbolic semantics of $(a, C)^*$ consists of all the symbolic words with n concatenated symbols $(a, C) \ldots (a, C)$. These symbolic words are in fact star-free sum-free ϵ-regular expressions. We can do the same for ECA by defining the *symbolic language* accepted by the ECA as the set of words over Ω which are the concatenation of the labels on some run, or, equivalently, the set of words over Ω which are accepted by the ECA when it is regarded as a finite automaton over Ω. The classical Kleene theorem then says that the classes of symbolic semantics of ϵ-regular expressions and of symbolic languages of ECA are equal. Note how the empty expression plays the role of the empty symbolic word.

Then two more things are left to prove: the first is that the concrete semantics of ϵ-regular expressions equals the union of the concrete semantics of all symbolic words in the symbolic semantics of the ϵ-regular expression. The second is that the limited observation language of an ECA equals the union of the concrete semantics of all symbolic words in the symbolic language of the ECA. Both proofs are straightforward.

Theorem 2 (Abstract Kleene theorem). *The class of sets of TSS which are the abstract language of some ECA equals the class of sets of TSS which are the abstract semantics of some ϵ-regular expression.*

This is a corollary of Theorem 1 and Proposition 1:

Note that similarily to all the above we may aprove that ϵ-free ECA are equivalent to regular expressions.

5 Conclusions

We have described here a class of regular expressions that have the same expressive power as event-clock automata. The work can easily be extended to automata on infinite (extended) timed state sequences and ω-regular expressions as in the untimed case [Tho90] or the timed automata case [ACM97]. Our result relies on a semantics of concatenation of regular expressions as a *partial operation* on a suitable defined class of timed state sequences. We think this happens as event-clock automata are an interleaving model of distributed real-time systems.

Recently Bouyer and Petit [BP99] have proved a Kleene theorem for timed automata that does not require renaming, as in [ACM97]. The problem seems to come again from defining concatenation on real-time items. In [BP99] this is solved by defining *several* concatenations constrained by the set of clocks to be reset at the "chopping" point. It would be interesting to see if our approach (i.e. keeping a unique, but partial, concatenation while changing the items to be concatenated) can be applied to timed automata too.

References

[AD94] R. Alur and D.L. Dill. A theory of timed automata, *Theoretical Computer Science*, 126, 183-235, 1994.

[AFH94] R. Alur, L. Fix and T.A. Henzinger. A determinizable class of timed automata, *Theoretical Computer Science*, 211, 253-273, 1999.

[ACM97] E. Asarin, P. Caspi and O. Maler, A Kleene Theorem for Timed Automata, in G. Winskel (Ed.) *Proc. LICS'97*, 160-171, 1997.

[BP99] P. Bouyer and A. Petit, Decomposition and composition of timed automata, to appear in *Proc. of ICALP '99*, LNCS series, 1999.

[Di1] C. Dima. Real-time automata and their class of accepted languages, draft of a UNU/IIST report, available at `http://funinf.math.unibuc.ro/~cdima/work/expressive.ps.gz`.

[Di2] C. Dima. Complementation of real-time automata, submitted to *FCT&TCS'99*, available at `http://funinf.math.unibuc.ro/~cdima/work/mfcs.ps.gz`.

[Di3] C. Dima. Automata and Regular Expressions for Real-Time Languages, submitted to *AFL'99*, abstract available at `http://funinf.math.unibuc.ro/~cdima/work/abstract.ps.gz`.

[Di4] C. Dima. Removing ϵ-transitions from event-clock automata, in preparation, available at `http://funinf.math.unibuc.ro/~cdima/work/eps-elim.ps.gz`.

[HRS98] T.A. Henzinger, J.-F. Raskin and P.-Y. Schobbens. The regular real-time languages, in *Proceedings of the 25-th International Colloquium on Automata, Languages and Programming* LNCS series, Springer Verlag, 1998.

[HU] John E. Hopcroft and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley/Narosa Publishing House, eighth edition, New Delhi, 1992.

[RS97] J.-F. Raskin and P.-Y. Schobbens. State-clock logic: a decidable real-time logic, in *Hybrid and Real-Time Systems*, LNCS 1201, 33-47, Springer Verlag, 1997.

[Tho90] W. Thomas. Automata on infinite objects, in J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, vol B, 133-191, Elsevier, Amsterdam, 1997.