Timed Regular Expressions

EUGENE ASARIN, PAUL CASPI, AND ODED MALER

VERIMAG, *Gières, France*

Abstract. In this article, we define *timed regular expressions*, a formalism for specifying discrete behaviors augmented with timing information, and prove that its expressive power is equivalent to the *timed automata* of Alur and Dill. This result is the timed analogue of Kleene Theorem and, similarly to that result, the hard part in the proof is the translation from automata to expressions. This result is extended from finite to infinite (in the sense of Büchi) behaviors. In addition to these fundamental results, we give a clean algebraic framework for two commonly accepted formalisms for timed behaviors, time-event sequences and piecewise-constant signals.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: *real-time and embedded systems*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*algebraic language theory*; *classes defined by grammars or automata*

General Terms: Languages, Theory

Additional Key Words and Phrases: Kleene theorem, timed automata, timed languages

1. *Introduction*

The theory of automata, by now about half a century old, constitutes the foundation for many branches in Computer Science. In essence, it is a theory about *sequences* of discrete events occurring one *after* the other and about formalisms for describing sets of such sequences, most notably by finite-state transition systems (automata) that generate or accept them. Since automata can model computer programs, digital circuits and many other discrete-event dynamical systems, they can be used for simulation, verification and synthesis of such systems.

Classical automata theory deals only with a *qualitative* notion of time: a sequence of events specifies the *ordering* of their occurrence times, but not the *distance* between them in terms of "real" time. While this level of abstraction has proven to be very useful for the analysis of certain systems, many application domains require more detailed models that include timing information. For example, we might want to refine a specification of the form "every a is followed by b" into "every a is followed by b *within* 5 *seconds*." Likewise, we might want to augment automaton

Authors' address: VERIMAG, Centre Equation, 2 av. de Vignate 38610, Gières, France, e-mail: {Eugene.Asarin;Paul.Caspi;Oded.Maler}@imag.fr.

models of systems with information concerning the time it takes to complete a transition. To this end a timed theory of automata and sequential behaviors needs to be developed, in which timed extensions of the ingredients of the classical theory can be investigated.

Timed automata [Alur and Dill 1994], automata equipped with clocks, have been studied extensively in recent years as they provide a rigorous model for reasoning about quantitative time. Together with other formalisms such as real-time logics, real-time process algebras and timed Petri nets, they constitute an underlying theoretical basis for the specification and verification of real-time systems. The main attraction of timed automata is due to their suitability for modeling certain time-dependent phenomena, and the decidability of their reachability (or empty language) problem, a fact that has been exploited in several verification tools, for example, Kronos [Yovine 1997] and Uppaal [Larsen et al. 1997].

On the theoretical front, however, the results are somewhat less satisfactory. The classical theory of automata is extremely simple and elegant. It establishes, for example, that the expressive power of finite automata is equivalent to that of a plethora of other formalisms such as *regular expressions, monadic second-order logic, linear language equations, rational formal series, finite monoids* as well as *sequential digital circuits*. Almost none of these facts has been proven for the general class of timed automata.[1]

In this article, we try to follow the spirit of Trakhtenbrot [1995], where a call was formulated to “lift” the classical results of automata theory to deal with timed automata. We investigate a timed version of one of the cornerstones of the classical theory, namely Kleene Theorem, which states that the *recognizable* sets (those accepted by finite nondeterministic automata) are exactly the *regular* (or *rational*) sets (those definable by regular expressions). An infinitary version of this theorem shows that regular sets of infinite sequences are exactly those recognized by Büchi ω-automata [Büchi 1960; McNaughton 1966]. To prove the timed analogues of these results we define *timed regular* and *timed ω-regular* expressions and show that they denote exactly what timed automata can recognize. As in the classical theorem one direction, the construction of automata from expressions, is rather straightforward, while the proof of the other direction, from automata to expressions, is much more involved. In order to match the expressive power of timed automata we use expressions that employ, in addition to the standard operators and a time-specific operator, two additional constructs, namely, *intersection* and *renaming*. In the preliminary version of this article, [Asarin et al. 1997] we have proved the necessity of intersection and conjectured the necessity of renaming—a fact proved later by Herrmann [1999]. The idea of using regular expressions to represent the behavior of hybrid systems (for which timed automata are a special case) was developed independently by Li et al. [1998] who proposed a formalism called *hybrid regular expressions*, to which some very restricted classes of hybrid automata can be translated. Other related formalisms and results by Bouyer and

[1] In fact, already in Alur and Dill [1994] it was proved that the class of languages accepted by timed automata is not closed under complementation and hence no simple *logical* characterization of this class exists.

Petit [1999, 2002] are discussed in Section 8. The rest of the article is organized as follows:

Section 2. We discuss two commonly-used models for timed behaviors, namely time-event sequences and piecewise-constant signals, and show how they can be obtained by combining the free monoid $(\Sigma^*, \cdot, \varepsilon)$ of event sequences with the commutative monoid $(\mathbb{R}_+, +, 0)$ of time passage. This short algebraic excursion can be skipped by those who can live without it.

Section 3. We introduce the syntax of timed regular expressions. The main novelty with respect to classical expressions is in the use of the time restriction operator $\langle\varphi\rangle_{[l,u]}$ that restricts the time-event sequences in φ to be of metric length in the interval $[l, u]$. Several classes of these expressions are introduced and relations between them are explored. In particular, the proof that the special $\circ$ and $^{\circledast}$ operators, which correspond to nonresetting automaton transitions, can be eliminated from expressions is an important contribution to the understanding of timed behaviors.

Section 4. Timed automata as acceptors of sets of finite time-event sequences are defined.

Section 5. The easy part of the timed Kleene Theorem, the transformation of expressions into timed automata is proved.

Section 6. In this section, we prove the harder direction of the main result, the translation of timed automata into expressions. We first remind the readers of the language equations used to prove the classical Kleene Theorem, and explain the difficulty in applying them to timed automata. Then we prove a useful lemma, stating that any language accepted by a timed automaton can be written as a *morphic image* of a *finite intersection* of languages accepted by *one-clock* timed automata. This allows us to do the rest of the proof using one-clock automata, which are relatively simpler. The one-clock automaton is transformed into a system of *quasi-linear language equations* which is solved using a variant of Gaussian elimination (these equations were first defined in Asarin [1998]). Collecting everything together we obtain our main result—Kleene Theorem for timed automata.

Section 7. We move on to infinite time-event sequence, define timed ω-regular expressions and timed ω-automata, and prove the correspondence between them (Büchi–McNaughton Theorem).

Section 8. We summarize the results and compare them with related work.

2. *Monoids, Event Sequences and Signals*

2.1. THE MONOIDS Σ^* AND $\mathbb{R}_+$. There are two basic approaches for enriching sequential discrete behaviors with metric timing information, one is, so to speak, event-based and the other is state-based.

—*Time-event sequences*. These are sequences where nonnegative time durations are inserted between events. Time-event sequences allow two events to happen at the *same* metric time instant (without any time passage between them) but still one *after* the other in the discrete sense. Time-event sequences are equivalent to the commonly used *timed traces* in which a nondecreasing sequence of *time stamps* is attached to an event sequence.

—*Signals*. Similarly to sequences that can be viewed as functions from an initial segment of $\mathbb{N}$ to an alphabet Σ, signals are functions from an initial segment $[0, r)$ of the non-negative real line $\mathbb{R}_+$ to Σ satisfying some additional sanity condition, for example, $[0, r)$ can be decomposed into a finite number of left-closed right-open intervals such that the value of the signal is constant on each interval. Such piecewise-constant signals are used extensively in modeling the behavior of digital circuits and in the presentation of solutions to scheduling problems.

In order to cast these objects in an algebraic framework, we need to consider the algebraic characterization of their two components, *discrete events* and *time passage*, and then mix them together.

A *monoid* is a triple $(M, \diamond, e)$ where M is a set, $\diamond$ is an associative binary operation on M and e is the identity element of M satisfying $e \diamond m = m \diamond e = m$ for every $m \in M$. The set of all finite sequences of elements taken from a set Σ is a monoid under the concatenation operation $\cdot$ and the empty word ε is its identity element. Such a monoid is called the *free monoid generated by* Σ and is denoted by $(\Sigma^*, \cdot, \varepsilon)$, or Σ^* for short. Note that Σ need not be finite nor countable: we can define, for example, $\mathbb{R}^*$ as the monoid of all finite sequences of real numbers. The free monoid is the primary object for describing behaviors of discrete-event systems and its subsets are the subject matter of formal language theory. We sometimes write $m_1 m_2$ instead of $m_1 \diamond m_2$ or $m_1 \cdot m_2$.

If we express the passage of time using *numbers*, then the significant operation is *addition*: if r_1 seconds pass and then additional r_2 seconds pass, the total elapsed time is $r_1 + r_2$ seconds. Sets such as $\mathbb{N}$, $\mathbb{Q}_+$ or $\mathbb{R}_+$ are monoids under addition, with 0 serving as the identity element. It is worth mentioning that they are commutative, that is, they satisfy $m_1 + m_2 = m_2 + m_1$. We concentrate on the more general monoid $(\mathbb{R}_+, +, 0)$ for which $\mathbb{N}$ and $\mathbb{Q}_+$ are submonoids.

2.2. MIXING MONOIDS. We want to create a monoid, whose elements consist of an *interleaving* of time passages and events (or of time passages of different sorts, when we consider signals). We use the following construction which allows to put elements of two monoids in a sequence:

The *free shuffle* of two monoids $(A, \diamond_a, e_a)$ and $(B, \diamond_b, e_b)$ is the monoid $M = (A \uplus B)^*$, namely the free monoid generated by the disjoint union of both A and B. An element of M may look like this:

$$a_1 \cdot a_2 \cdot b_1 \cdot e_a \cdot b_2 \cdot a_3 \cdot e_b \cdot b_3 \tag{1}$$

In order to obtain a *canonical form*, in which there is always an *alternation* of elements of the two monoids, we define a congruence relation[2] generated by the following equalities:

$$\begin{aligned} a_i \cdot a_j &= a_i \diamond_a a_j \\ b_i \cdot b_j &= b_i \diamond_b b_j \\ e_a &= e_b = \varepsilon. \end{aligned} \tag{2}$$

[2] A congruence is an equivalence relation $\sim$, which is closed under the monoid operation, that is $m \sim m'$ implies $m_1 \cdot m \cdot m_2 \sim m_1 \cdot m' \cdot m_2$ for every $m_1, m_2 \in M$.

These rules allow to replace two adjacent elements in the sequence, which come from the same monoid, by one element, and to get rid of "dummy" identity elements. Applying these rules, we can reduce any element of an equivalence class into a canonical form which is an alternating sequence of elements of A and B. For example, the sequence in (1) can be reduced to

$$(a_1 \diamond_a a_2) \cdot (b_1 \diamond_b b_2) \cdot a_3 \cdot b_3$$

We call $\sim$ the reduction congruence on $(A \uplus B)^*$. The set of congruence classes of $\sim$, also known as the quotient $M/\sim$, is a monoid as well. This is a well-known construction on monoids (see Howie [1995]) and on algebraic structures in general:

Definition 2.1 (*Free Products of Monoids*). Let $(A, \diamond_a, e_a)$ and $(B, \diamond_b, e_b)$ be two monoids. Their free product is $A \boxplus B = (A \uplus B)^*/\sim$ where $\sim$ is the reduction congruence.

The properties of $A \boxplus B$ can be described in a category-theoretic setting, where it is termed the *co-product* of A and B. There are two canonical morphisms $i_a : A \to A \boxplus B$ and $i_b : B \to A \boxplus B$ which insert elements of A and B respectively into $A \boxplus B$. Any pair of morphisms $\theta_a : A \to C$, and $\theta_b : B \to C$ to a third monoid C, induces a morphism $\theta = \theta_a \boxplus \theta_b$ from $A \boxplus B$ to C (the co-product of θ_a and θ_b), as can be visualized by the following commutative diagram:

$$\begin{tikzcd}
A \arrow[r, "i_a"] \arrow[dr, "\theta_a"'] & A \boxplus B \arrow[d, "\theta"] & B \arrow[l, "i_b"'] \arrow[dl, "\theta_b"] \\
& C &
\end{tikzcd}$$

In particular, to project $A \boxplus B$ onto A, let θ_a be the identity $Id_a : A \to A$ and let θ_b be the constant function $e_a : B \to A$ which maps B to the identity element of A. This way we obtain the canonical projection $\pi_a : A \boxplus B \to A$:

$$\begin{tikzcd}
A \arrow[r, "i_a"] \arrow[dr, "Id_a"'] & A \boxplus B \arrow[d, "\pi_a"] & B \arrow[l, "i_b"'] \arrow[dl, "e_a"] \\
& A &
\end{tikzcd}$$

2.3. Time-Event Sequences

Definition 2.2 (*The Time-Event Monoid*). The time-event monoid over a set Σ of events is the free product $\mathcal{T}(\Sigma) = \Sigma^* \boxplus \mathbb{R}_+$ of the free monoid over Σ and the monoid of nonnegative real numbers under addition.

When the alphabet Σ is clear from the context we will use $\mathcal{T}$ instead of $\mathcal{T}(\Sigma)$. A typical element of the free shuffle will look like:

$$0.7 \cdot a \cdot b \cdot 3 \cdot 5.4 \cdot ab \cdot c \cdot 0 \cdot a \cdot \varepsilon \cdot 5.4 \cdot a \cdot 0.2$$

and after reduction into canonical form as:

$$0.7 \cdot ab \cdot 8.4 \cdot abca \cdot 5.4 \cdot a \cdot 0.2.$$

For completeness sake, we mention that as a timed trace, this sequence (without the last term 0.2) will be written as:

$$(a, 0.7), (b, 0.7), (a, 9.1), (b, 9.1), (c, 9.1), (a, 9.1), (a, 14.5).$$

Time-event sequences seem to be conceptually clearer than timed traces as the same type of concatenation applies to events and time durations. The philosophy behind time-event sequences is the one employed in the timed automata literature: a behavior is an alternating sequence of time passages and of events, which occur at certain time points and consume no time. There are two natural projections on $\mathcal{T}$, one that ignores the events and one that ignores the metric information:

Definition 2.3 (*Untime and Length*). Let $\mathcal{T} = \Sigma^* \boxplus \mathbb{R}_+$

—The length morphism $\lambda : \mathcal{T} \to \mathbb{R}_+$ is the projection on $\mathbb{R}_+$ obtained by mapping elements of Σ^* to 0.

—The untime morphism $\mu : \mathcal{T} \to \Sigma^*$ is the projection on Σ^* obtained by mapping elements of $\mathbb{R}_+$ to ε.

Clearly, $\lambda(u)$ is the *duration* of the time-event sequence u, while $\mu(u)$ is the sequence of all the discrete events in u without timing information. For example:

$$\lambda(0.7 \cdot ab \cdot 8.4 \cdot abca \cdot 5.4 \cdot a \cdot 0.2) = 14.7$$

and

$$\mu(0.7 \cdot ab \cdot 8.4 \cdot abca \cdot 5.4 \cdot a \cdot 0.2) = ababcaa.$$

In this article, we use $\mathcal{T}$ as the underlying set for timed languages on which we prove Kleene theorem. For the sake of completeness we will formalize below the equally important and intuitive concept of continuous-time, piecewise-constant signals. The appropriate timed automata for accepting signals were described in Asarin et al. [1997] along with a proof of their corresponding Kleene theorem.

2.4. SIGNALS. The main difference between signals and time-event sequences is that in signals discrete values are associated directly with time durations: a signal may have one value inside a time interval of length r_1, then another value for a duration of r_2, etc. This motivates the idea of *multi-sorted time* formalized as follows.

Definition 2.4 (*The Signal Monoid*). Let Σ be an m-element set, and let $\{{}_a\mathbb{R}_+ : a \in \Sigma\}$ be m distinct copies of the monoid $\mathbb{R}_+$. The signal monoid over Σ is the free product $\mathcal{S}(\Sigma) = \boxplus_{a\in\Sigma}\, {}_a\mathbb{R}_+$.

It is convenient to use exponential notation for elements of ${}_a\mathbb{R}_+$. For example, $3.2 \in {}_b\mathbb{R}_+$ can be written as $b^{3.2}$ and read as "b during 3.2 time units." Using this notation, a typical element of the free shuffle for $\Sigma = \{a, b, c\}$ would be

$$a^5 \cdot b^2 \cdot b^{4.2} \cdot a^{2.5} \cdot b^0 \cdot c^7$$

whose normal form after reduction is

$$a^5 \cdot b^{6.2} \cdot a^{2.5} \cdot c^7.$$

Two features distinguish signals from time-event sequences:

(1) *Filtering of Zero-Duration Events*. With signals, it is impossible to express a phenomenon such as "the signal value was a for some time, then switched to b and then *immediately* to c" because of the elimination of b^0. This conforms to the usual semantic interpretation of signals as *functions* from $\mathbb{R}_+$ to Σ, which have a unique value at every time instant.[3]

(2) *Stuttering*. Two consecutive elements a^r and a^s are reduced in the normal form to a^{r+s}. Hence, the untiming of a signal should be a *nonstuttering* sequence (a sequence without two consecutive occurrences of the same letter) or, equivalently, the stuttering closure of such a sequence.

In order to define the untiming of signals we need to introduce the *stuttering-closed monoid* generated by Σ, which is $\Sigma^{\heartsuit} = \Sigma^*/\approx$, where $\approx$ is the congruence generated by the equalities of the form

$$aa = a$$

for every $a \in \Sigma$. Hence, a sequence such as $abac$ stands for the equivalence class $a^+b^+a^+c^+$.

Definition 2.5 (*Untime and Length for Signals*). Let $\mathcal{S}(\Sigma) = \boxplus_{a\in\Sigma} {}_a\mathbb{R}_+$

—The length morphism $\lambda : \mathcal{S} \to \mathbb{R}_+$ is obtained as a co-product of m morphisms of the form $\theta_a : {}_a\mathbb{R}_+ \to \mathbb{R}_+$.

—The untime morphism $\mu : \mathcal{S} \to \Sigma^{\heartsuit}$ is obtained as a co-product of m morphisms of the form $\theta_a : {}_a\mathbb{R}_+ \to \Sigma^{\heartsuit}$, which map a^0 to ε and a^r (with $r > 0$) to a.

The reader can verify that these are the intuitive meanings of length and qualitative behavior associated with signals. For example,

$$\lambda(a^5 \cdot b^{6.2} \cdot a^{2.5} \cdot c^7) = 20.7$$

and

$$\mu(a^5 \cdot b^{6.2} \cdot a^{2.5} \cdot c^7) = abac.$$

The framework of mixing monoids allows to define easily an algebraic structure for the most general situation where both piecewise-constant behaviors and discrete events can occur in the same system. For completeness, we give a definition:

Definition 2.6 (*Signal-Event Monoid*). Let Σ_1 and Σ_2 be finite sets (signal alphabet and events alphabet). Let ${}_a\mathbb{R}_+, a \in \Sigma_1$ be distinct copies of the monoid $\mathbb{R}_+$. The signal-events monoid over Σ_1, Σ_2 is the free product $\mathcal{ST}(\Sigma_1, \Sigma_2) = \boxplus_{a\in\Sigma_1} {}_a\mathbb{R}_+ \boxplus \Sigma_2^*$.

For example, for $\Sigma_1 = \{a, b, c\}$ and $\Sigma_2 = \{x, y, z\}$, a typical element of the signal-event monoid would be

$$a^5 \cdot xy \cdot b^{6.2} \cdot a^{2.5} \cdot z \cdot c^7 \cdot y.$$

[3] If zero durations are not eliminated one has to resort to constructs such as "super-dense" lexicographically ordered time in order to maintain the notion of a behavior as a function from time to states, see, for example Maler et al. [1992].

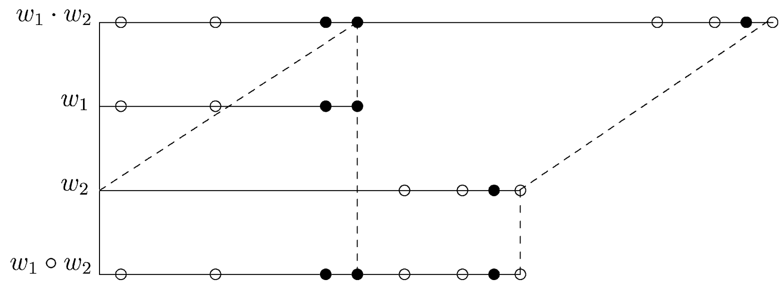

FIG. 1. Two concatenation operations.

2.5. TIMED LANGUAGES AND OPERATIONS. From now on, we restrict ourselves to the monoid $\mathcal{T}$ of time-event sequences and its subsets which we call *timed languages*. We denote the concatenation operation by $\cdot$, and define an additional concatenation operation, specific to timed languages. Before introducing the syntax we need some preliminary definitions.

Definition 2.7 (*Left Derivative*). For every two sequences u and v, the left derivative of u by v is a partial function defined as:

$$v \backslash u = \begin{cases} w & \text{if } \exists w \; u = vw \\ \perp & \text{otherwise.} \end{cases}$$

In other words, $v \backslash u$ is defined if v is a prefix of u, and in that case v is removed.

Definition 2.8 (*Absorbing Concatenation*). The partial operator $\circ$ on $\mathcal{T}$ is defined as:

$$u \circ v = u \cdot (\lambda(u) \backslash v)$$

that is, $u \circ v$ is defined only if v starts with a time duration of at least $\lambda(u)$, and in that case $\lambda(u)$ time is removed from the front of v before concatenation.

For example, $(a \cdot 5 \cdot b) \circ (3 \cdot c) = \perp$ and $(a \cdot 5 \cdot b) \circ (7 \cdot c) = a \cdot 5 \cdot b \cdot 2 \cdot c$. Note that $\lambda(u \circ v) = \lambda(v)$ whenever $u \circ v$ is defined. The $\circ$ operation is motivated, as we shall see later, by timed automaton transitions that *do not reset a clock*. This operation, similarly to concatenation, can be extended to an operation on timed languages by letting $L_1 \circ L_2 = \{u \circ v : u \in L_1 \wedge v \in L_2\}$. Figure 1 illustrates absorbing concatenation in comparison with the standard one.

In order to prove that languages accepted by timed automata can be expressed using timed regular expressions, we will need sometimes to split the alphabet of the automaton, define the expression on the extended alphabet and than map it back to the original alphabet using the following operation.

Definition 2.9 (*Renaming*). Let Σ_1 and Σ_2 be two alphabets. A renaming from Σ_1 to Σ_2 is a function $\theta : \Sigma_1 \to \Sigma_2 \cup \{\varepsilon\}$. We use the same symbol for the natural extensions of θ to sequences, $\theta : \Sigma_1^* \to \Sigma_2^*$, and time-event sequences, $\theta : (\Sigma_1^* \boxplus \mathbb{R}_+) \to (\Sigma_2^* \boxplus \mathbb{R}_+)$.

3. *Timed Regular Expressions*

An *integer-bounded interval* is either $[l, u]$, $(l, u]$, $[l, u)$, or (l, u) where $l \in \mathbb{N}$ and $u \in \mathbb{N} \cup \{\infty\}$ such that $l \leq u$. We exclude $\infty]$ and use l for $[l, l]$. In the following definition, we introduce several classes of regular expressions, each using another subset of the expression formation rules.

Definition 3.1 (*Timed Regular Expressions*). Timed regular expressions over an alphabet Σ (also referred to as Σ-expressions) are defined using the following families of rules.

(1) $\underline{a}$ for every letter $a \in \Sigma$ and the special symbol ε are expressions.
(2) If φ, φ_1 and φ_2 are Σ-expressions and I is an integer-bounded interval, then $\langle\varphi\rangle_I$, $\varphi_1 \cdot \varphi_2$, $\varphi_1 \vee \varphi_2$ and φ^* are Σ-expressions.
(3) If φ, φ_1 and φ_2 are Σ-expressions, then $\varphi_1 \circ \varphi_2$ and $\varphi^{\circledast}$ are Σ-expressions.
(4) If φ_1 and φ_2 are Σ-expressions, φ_0 is a Σ_0-expression for some alphabet Σ_0, and $\theta : \Sigma_0 \to \Sigma \cup \{\varepsilon\}$ is a renaming, then $\varphi_1 \wedge \varphi_2$ and $\theta(\varphi_0)$ are Σ-expressions.

Expressions formed using rules (1) and (2) are called timed regular expressions and denoted by $\mathcal{E}(\Sigma)$. If, in addition, rule (3) is applied we call them extended timed regular expression and denote them by $\mathcal{EE}(\Sigma)$. Rules (1), (2), and (4) yield generalized timed regular expressions denoted by $\mathcal{GE}(\Sigma)$. Finally, the generalized extended expressions ($\mathcal{GEE}$) are obtained using all the four rules.

The semantics of (generalized extended) timed regular expressions, $[\![\ ]\!]$: $\mathcal{GEE}(\Sigma) \to 2^{\mathcal{T}}$, is given by:

$$
\begin{array}{lcl}
[\![\varepsilon]\!] & = & \{\varepsilon\} \\
[\![\underline{a}]\!] & = & \{r \cdot a : r \in \mathbb{R}_+\} \\
[\![\langle\varphi\rangle_I]\!] & = & [\![\varphi]\!] \cap \{u : \lambda(u) \in I\} \\
[\![\varphi_1 \vee \varphi_2]\!] & = & [\![\varphi_1]\!] \cup [\![\varphi_2]\!] \\
[\![\varphi_1 \cdot \varphi_2]\!] & = & [\![\varphi_1]\!] \cdot [\![\varphi_2]\!] \\
[\![\varphi^*]\!] & = & \bigcup_{i=0}^{\infty}([\![\underbrace{\varphi \cdot \ldots \cdot \varphi}_{i \text{ times}}]\!]) \\
\hline
[\![\varphi_1 \circ \varphi_2]\!] & = & [\![\varphi_1]\!] \circ [\![\varphi_2]\!] \\
[\![\varphi^{\circledast}]\!] & = & \bigcup_{i=0}^{\infty}([\![\underbrace{\varphi \circ \ldots \circ \varphi}_{i \text{ times}}]\!]) \\
\hline
[\![\varphi_1 \wedge \varphi_2]\!] & = & [\![\varphi_1]\!] \cap [\![\varphi_2]\!] \\
[\![\theta(\varphi)]\!] & = & \{\theta(u) : u \in [\![\varphi]\!]\}
\end{array}
$$

The novel features here with respect to untimed regular expressions are the meaning of the atom $\underline{a}$ that represents an arbitrary passage of time followed by an event a and the $\langle\varphi\rangle_I$ operator that restricts the metric length of the time-event sequences in $[\![\varphi]\!]$ to be in the interval I. We show in the next section that the absorbing concatenation $\circ$ and the absorbing iteration $^{\circledast}$ can always be eliminated and hence timed regular expressions and extended timed regular expressions have the same expressive power. We call the corresponding class of languages *timed regular languages*. Unfortunately this class does not match the expressive power of timed automata, which requires both renaming and intersection.

We use the following shorthands:

$$a = \langle \underline{a} \rangle_0; \quad \varphi^+ = \varphi \cdot \varphi^*; \quad \varphi^{\oplus} = \varphi \circ \varphi^{\circledast}; \quad \varphi^{\circ i} = \underbrace{\varphi \circ \ldots \circ \varphi}_{i \text{ times}}.$$

Operations $\vee, \cdot$ and * satisfy well-known properties of Kleene algebra (see Conway [1971]). We state some simple additional algebraic properties involving absorbing concatenation.

PROPOSITION 3.2 (ALGEBRAIC PROPERTIES OF ABSORBING CONCATENATION). *The* $\circ$ *operation satisfies the following equalities*:

—$\vee$*-Distributivity.* $(\alpha \vee \beta) \circ \gamma = \alpha \circ \gamma \vee \beta \circ \gamma$ *and* $\alpha \circ (\beta \vee \gamma) = \alpha \circ \beta \vee \alpha \circ \gamma$
—*Associativity.* $(\alpha \circ \beta) \circ \gamma = \alpha \circ (\beta \circ \gamma)$
—*Mixed Associativity.* $\alpha \circ (\beta \cdot \gamma) = (\alpha \circ \beta) \cdot \gamma$ *if* $\beta \cap \mathbb{R}_+ = \emptyset$.[4]

The situation with mixed associativity is not as good as it could be: typically $\alpha \cdot (\beta \circ \gamma) \neq (\alpha \cdot \beta) \circ \gamma$.

We illustrate the semantics of the expressions and some obvious properties via examples. The first examples demonstrate the interaction between time restriction and standard concatenation. Let

$$\begin{aligned} \varphi_1 &= \langle \underline{a} \rangle_{[1,2]} \\ \varphi_2 &= \langle \underline{a} \rangle_{[1,2]} \cdot \langle \underline{b} \rangle_{[2,4]} \\ \varphi_3 &= \langle \underline{a} \cdot \underline{b} \rangle_{[3,6]} \end{aligned}$$

The semantics of these expressions is the following:

$$\begin{aligned} [\![\varphi_1]\!] &= \{r \cdot a : r \in [1,2]\} \\ [\![\varphi_2]\!] &= \{r_1 \cdot a \cdot r_2 \cdot b : r_1 \in [1,2] \wedge r_2 \in [2,4]\} \\ [\![\varphi_3]\!] &= \{r_1 \cdot a \cdot r_2 \cdot b : r_1 + r_2 \in [3,6]\} \end{aligned}$$

Expression φ_1 allows a to occur anywhere in the $[1,2]$ interval. Similarly φ_2 allows b to occur between 2 and 4 time units after the occurrence of a, while φ_3 constrains b to occur in the interval $[3,6]$ and after the occurrence of a. Clearly, $[\![\varphi_2]\!] \subseteq [\![\varphi_3]\!]$.

Putting time restriction outside the Kleene star, we can express constraints involving an *unbounded* number of time durations. The expression

$$\langle \underline{a}^* \rangle_{[1,2]}$$

denotes the set

$$\left\{ r_1 \cdot a \cdot r_2 \cdot a \cdots r_k \cdot a : k \in \mathbb{N} \wedge \sum_{i=1}^{k} r_i \in [1,2] \right\}.$$

[4] The meaning of this restriction is that every $u \in \beta$ contains at least one discrete event $a \in \Sigma$. It can be also written as $\varepsilon \notin \mu(\beta)$. Without this restriction, we have, for example, $5 \circ (3 \cdot 7) = 10$ while $(5 \circ 3) \cdot 7 = \emptyset$.

The role of intersection is to express "unbalanced parentheses" like in the expression

$$(\langle \underline{a} \cdot \underline{b} \rangle_3 \cdot \underline{c}) \wedge (\underline{a} \cdot \langle \underline{b} \cdot \underline{c} \rangle_3)$$

denoting the set

$$\{r_1 \cdot a \cdot r_2 \cdot b \cdot r_3 \cdot c : (r_1 + r_2 = 3) \wedge (r_2 + r_3 = 3)\}.$$

In Asarin et al. [1997], we showed that this language, recognizable by a simple timed automaton with two clocks, cannot be expressed without intersection (see also Herrmann [1999] for another proof).

The role of renaming in the translation from automata to expressions will be elaborated in Section 6.2. Using the syntax we have chosen for time-event sequences, it is impossible to express without renaming sets containing any time-event sequence that does not terminate with an event, that is, sequence of the form $w \cdot r$ such that $r > 0$. Using renaming, it can be expressed as the image of $w \cdot \langle \underline{a} \rangle_r$ where a is mapped to ε. Such time-event sequences can be expressed using a richer syntax which allows to specify arbitrary timed durations without events (we do not use them because they complicate other proofs). However, renaming remains necessary even for such a richer syntax (and for signals). The language

$$\left\{ r_1 \cdot a \cdots r_k \cdot a : 1 < j < k \quad \text{and} \quad \sum_{i=1}^{j} r_i = \sum_{i=j}^{k} r_i = 1 \right\},$$

over the alphabet $\{a\}$ can be expressed as the image of the $\{a, b\}$-language given by the expression

$$\langle \underline{a}^+ \cdot \underline{b} \rangle_1 \cdot \underline{a}^+ \wedge \underline{a}^+ \cdot \langle \underline{b} \cdot \underline{a}^+ \rangle_1$$

via the morphism $\theta : a \mapsto a, b \mapsto a$. It was proved in Herrmann [1999] that this language cannot be expressed without renaming, although it can be recognized by a timed automaton.

The rest of this section is devoted to the nonstandard $\circ$ and $^{\circledast}$ operations which facilitate the translation from automata to expressions but, as we show in the sequel, do not contribute to the expressive power of timed regular expressions. We start with some examples.

The $\circ$ operation acts like standard concatenation whenever the second operand denotes a language *without* a restriction on the duration of time *before* the *first* event. For example

$$\underline{a} \circ \underline{b} = \underline{a} \cdot \underline{b}.$$

On the other hand, consider the expression

$$\langle \underline{a} \rangle_{[1,4]} \circ \langle \underline{b} \rangle_{[2,3]}.$$

Using $\circ$ means that the time spent in $\langle \underline{a} \rangle_{[1,4]}$ is taken into account in $\langle \underline{b} \rangle_{[2,3]}$, which is equivalent to pushing the first subexpression inside the parentheses of the second to get the expression

$$\left\langle \langle \underline{a} \rangle_{[1,4]} \cdot \underline{b} \right\rangle_{[2,3]}$$

whose semantics is the set

$$\{r \cdot a \cdot s \cdot b : r \in [1, 4] \wedge r + s \in [2, 3]\}.$$

Since $r + s \le 3$ implies $r \le 3$, this is equivalent to the expression

$$\big\langle \langle \underline{a} \rangle_{[1,3]} \cdot \underline{b} \big\rangle_{[2,3]}.$$

In general, occurrences of the $\circ$ operation can be transformed into $\cdot$ by moving parentheses; however, the *first* time restriction of the second operand should be isolated and made explicit. In case that the second operand starts with an iteration, the first occurrence should be pulled out from the scope of *, for example:

$$\begin{aligned} \underline{a} \circ (\langle \underline{b} \rangle_5)^* &= \underline{a} \circ (\varepsilon \vee \langle \underline{b} \rangle_5 \cdot (\langle \underline{b} \rangle_5)^*) = \underline{a} \circ \varepsilon \vee \underline{a} \circ (\langle \underline{b} \rangle_5 \cdot (\langle \underline{b} \rangle_5)^*) \\ &= \langle \underline{a} \rangle_0 \vee (\underline{a} \circ \langle \underline{b} \rangle_5) \cdot (\langle \underline{b} \rangle_5)^* = \langle \underline{a} \rangle_0 \vee \langle \underline{a} \cdot \underline{b} \rangle_5 \cdot (\langle \underline{b} \rangle_5)^*. \end{aligned}$$

The case of $^{\circledast}$ is more complicated. Consider the expression

$$\big(\langle \underline{a} \rangle_{[1,3]}\big)^{\circledast} = \bigvee_{i=0}^{\infty} \big(\langle \underline{a} \rangle_{[1,3]}\big)^{\circ i}$$

and take one of the components of the infinite union

$$\big(\langle \underline{a} \rangle_{[1,3]}\big)^{\circ 4} = \langle \underline{a} \rangle_{[1,3]} \circ \langle \underline{a} \rangle_{[1,3]} \circ \langle \underline{a} \rangle_{[1,3]} \circ \langle \underline{a} \rangle_{[1,3]},$$

which, by pushing parentheses, can be rewritten as

$$\big\langle \big\langle \big\langle \langle \underline{a} \rangle_{[1,3]} \cdot \underline{a} \big\rangle_{[1,3]} \cdot \underline{a} \big\rangle_{[1,3]} \cdot \underline{a} \big\rangle_{[1,3]}.$$

The corresponding semantics is

$$\begin{aligned} \{r_1 \cdot a \cdot r_2 \cdot a \cdot r_3 \cdot a \cdot r_4 \cdot a :\ & r_1 \in [1,3] \wedge \\ & r_1 + r_2 \in [1,3] \wedge \\ & r_1 + r_2 + r_3 \in [1,3] \wedge \\ & r_1 + r_2 + r_3 + r_4 \in [1,3]\}. \end{aligned}$$

As one can see, the first and last inequalities imply, due to convexity, the other “internal” inequalities and thus

$$\big(\langle \underline{a} \rangle_{[1,3]}\big)^{\circ 4} = \big\langle \langle \underline{a} \rangle_{[1,3]} \cdot \underline{a} \cdot \underline{a} \cdot \underline{a} \big\rangle_{[1,3]}$$

and, more generally

$$\big(\langle \underline{a} \rangle_{[1,3]}\big)^{\circledast} = \varepsilon \vee \big\langle \langle \underline{a} \rangle_{[1,3]} \cdot \underline{a}^* \big\rangle_{[1,3]}.$$

The convexity argument is the main idea behind the elimination of $^{\circledast}$. Due to the additivity of time it is sufficient to test the length after the first occurrence (for the lower-bound) and the last occurrence (for the upper-bound). For the occurrences in between, we can apply * to an “untimed” version of the expression without worrying. The next two examples demonstrate the special role of timing bounds appearing at the *beginning* of the expression under $^{\circledast}$.

Consider first the expression

$$(\langle \underline{a} \rangle_I \cdot \langle \underline{b} \rangle_J)^{\circledast}$$

for some intervals I and J. In this case,

$$(\langle \underline{a} \rangle_I \cdot \langle \underline{b} \rangle_J)^{\circ 3} = (\langle \underline{a} \rangle_I \cdot \langle \underline{b} \rangle_J) \circ (\langle \underline{a} \rangle_I \cdot \langle \underline{b} \rangle_J) \circ (\langle \underline{a} \rangle_I \cdot \langle \underline{b} \rangle_J)$$

and by pushing parentheses we get the expression

$$\langle\langle\langle\underline{a}\rangle_I \cdot \langle\underline{b}\rangle_J \cdot \underline{a}\rangle_I \cdot \langle\underline{b}\rangle_J \cdot \underline{a}\rangle_I \cdot \langle\underline{b}\rangle_J$$

whose semantics is:

$$\begin{aligned}\{r_1 \cdot a \cdot s_1 \cdot b \cdot r_2 \cdot a \cdot s_2 \cdot b \cdot r_3 \cdot a \cdot s_3 \cdot b :\ & r_1 \in I \wedge \\ & s_1 \in J \wedge \\ & r_1 + s_1 + r_2 \in I \wedge \\ & s_2 \in J \wedge \\ & r_1 + s_1 + r_2 + s_2 + r_3 \in I \wedge \\ & s_3 \in J\}.\end{aligned}$$

Here the convexity argument applies only to $\langle\underline{a}\rangle_I$ and the length of each and every $\underline{b}$ should be in J:

$$(\langle\underline{a}\rangle_I \cdot \langle\underline{b}\rangle_J)^{\circledast} = \varepsilon \vee \langle\langle\underline{a}\rangle_I \cdot (\langle\underline{b}\rangle_J \cdot \underline{a})^*\rangle_I \cdot \langle\underline{b}\rangle_J.$$

On the other hand, in the expression

$$(\langle\underline{a}\rangle_I \circ \langle\underline{b}\rangle_J)^{\circledast} = (\langle\langle\underline{a}\rangle_I \cdot \underline{b}\rangle_J)^{\circledast}$$

both $\underline{a}$ and $\underline{b}$ are in the scope of timing restrictions that appear at the beginning of the expression. Taking

$$(\langle\langle\underline{a}\rangle_I \cdot \underline{b}\rangle_J)^{\circ 3} = (\langle\langle\underline{a}\rangle_I \cdot \underline{b}\rangle_J) \circ (\langle\langle\underline{a}\rangle_I \cdot \underline{b}\rangle_J) \circ (\langle\langle\underline{a}\rangle_I \cdot \underline{b}\rangle_J)$$

and pushing all the parentheses forward, we obtain

$$\langle\langle\langle\langle\langle\langle\underline{a}\rangle_I \cdot \underline{b}\rangle_J \cdot \underline{a}\rangle_I \cdot \underline{b}\rangle_J \cdot \underline{a}\rangle_I \cdot \underline{b}\rangle_J.$$

The semantics of this expression is:

$$\begin{aligned}\{r_1 \cdot a \cdot s_1 \cdot b \cdot r_2 \cdot a \cdot s_2 \cdot b \cdot r_3 \cdot a \cdot s_3 \cdot b :\ & r_1 \in I \wedge \\ & r_1 + s_1 \in J \wedge \\ & r_1 + s_1 + r_2 \in I \wedge \\ & r_1 + s_1 + r_2 + s_2 \in J \wedge \\ & r_1 + s_1 + r_2 + s_2 + r_3 \in I \wedge \\ & r_1 + s_1 + r_2 + s_2 + r_3 + s_3 \in J\}.\end{aligned}$$

As before, only the first two and the last two inequalities are informative and the rest are redundant:

$$(\langle\langle\underline{a}\rangle_I \cdot \underline{b}\rangle_J)^{\circledast} = \varepsilon \vee \langle\langle\underline{a}\rangle_I \cdot \underline{b}\rangle_J \vee \langle\langle\langle\langle\underline{a}\rangle_I \cdot \underline{b}\rangle_J (\underline{ab})^* \cdot \underline{a}\rangle_I \cdot \underline{b}\rangle_J.$$

This is the intuition underlying the fact that $\circ$ and $^{\circledast}$ can be eliminated altogether. The proof of this fact will use induction on the *weight* of the regular expression, which, informally speaking, denotes the number of $\langle\cdot\rangle_I$ operations appearing at the "front" of the expression, i.e. in the sub-expressions that denote the beginning of the time-event sequences in the corresponding language.

Definition 3.3 (*Weight of a Regular Expression*). The weight is a function $\zeta : \mathcal{EE} \to \mathbb{N}$ defined inductively as:

$$\begin{aligned}\zeta(\underline{a}) &= 0 \\ \zeta(\varepsilon) &= 0 \\ \zeta(\delta_1 \vee \delta_2) &= \zeta(\delta_1) + \zeta(\delta_2)\end{aligned}$$

$$\begin{aligned}
\zeta(\delta_1 \cdot \delta_2) &= \begin{cases} \zeta(\delta_1) + \zeta(\delta_2) & \text{if } \varepsilon \in [\![\delta_1]\!] \\ \zeta(\delta_1) & \text{if } \varepsilon \notin [\![\delta_1]\!] \end{cases} \\
\zeta(\delta^*) &= \zeta(\delta) \\
\zeta(\langle\delta\rangle_I) &= \zeta(\delta) + 1.
\end{aligned}$$

The rule for $\delta_1 \vee \delta_2$ is due to the fact that its front consists of the fronts of δ_1 and δ_2. If δ_1 contains ε then δ_2 is part of the front of $\delta_1 \cdot \delta_2$. The rule for δ^* (for $\delta \not\ni \varepsilon$) follows from the identity $\delta^* = \delta \cdot \delta^* \vee \varepsilon$. It should be noted that the weight is a measure on the syntax and not on the semantics: $\langle\delta_1 \vee \delta_2\rangle_I$ has a smaller weight than $\langle\delta_1\rangle_I \vee \langle\delta_2\rangle_I$ although they are equivalent.

As usual in the theory of formal languages, special attention should be paid to the membership of ε in a given language (e.g., testing this membership is needed in order to compute the weight function). The next lemma allows to test this membership and to remove ε when necessary without changing the weight.

LEMMA 3.4 (TESTING AND REMOVING ε). *For a timed regular expression γ, it can be effectively tested whether or not $\varepsilon \in [\![\gamma]\!]$. An expression $\nu(\gamma)$ such that $[\![\nu(\gamma)]\!] = [\![\gamma]\!] - \{\varepsilon\}$ can be effectively constructed. The operation ν preserves the weight.*

Both a Boolean-valued function τ testing whether $\varepsilon \in \gamma$ and the operator ν (removing ε) can be defined recursively as follows.

$$\begin{aligned}
\tau(\underline{a}) &= 0 & \nu(\underline{a}) &= \underline{a} \\
\tau(\varepsilon) &= 1 & \nu(\varepsilon) &= \emptyset \\
\tau(\delta_1 \vee \delta_2) &= \tau(\delta_1) \vee \tau(\delta_2) & \nu(\delta_1 \vee \delta_2) &= \nu(\delta_1) \vee \nu(\delta_2) \\
\tau(\delta_1 \cdot \delta_2) &= \tau(\delta_1) \wedge \tau(\delta_2) & \nu(\delta_1 \cdot \delta_2) &= \begin{cases} \nu(\delta_1) \cdot \delta_2 \vee \nu(\delta_2) & \text{if } \tau(\delta_1) = 1 \\ \delta_1 \cdot \delta_2 & \text{if } \tau(\delta_1) = 0 \end{cases} \\
\tau(\delta_1^*) &= 1 & \nu(\delta_1^*) &= \nu(\delta_1) \cdot \delta_1^* \\
\tau(\langle\delta_1\rangle_I) &= \tau(\delta_1) \wedge (0 \in I) & \nu(\langle\delta_1\rangle_I) &= \langle\nu(\delta_1)\rangle_I
\end{aligned}$$

PROOF. We leave the proof of weight-preservation to the reader. □

The next result gives a characterization of expressions of weight 0 and a single weight-increasing rule allowing to obtain any regular language. We call expressions of the form $\bigvee_i \underline{a_i} \cdot \varphi_i$ *slow expressions*—in these expressions (whose weight is zero) there is no upper-bound on the occurrence time of the first event. An expression is *ε-free* if its semantics does not contain ε.

LEMMA 3.5 (SPECIAL FORM OF EXPRESSIONS)

(1) *Any expression of weight* 0 *is equivalent either to γ or to $\gamma + \varepsilon$ where γ is a slow expression.*

(2) *Any expression γ of a nonzero weight can be rewritten as*

$$\gamma = \langle\alpha\rangle_I \cdot \varphi \vee \beta, \tag{3}$$

where $\alpha, \beta, \varphi \in \mathcal{E}$ (or β is empty), α is ε-free, and $\zeta(\gamma) = \zeta(\alpha) + \zeta(\beta) + 1$.

In other words, this lemma says that starting from slow expressions and using only the inductive rule (3), we can build expressions for all regular languages. The proofs of the two statements are similar and we prove here only the second, more complicated one.

PROOF. The idea of the proof is simple: since $\zeta(\gamma) > 0$, γ is not atomic and there is at least one $\langle\rangle_I$ operator in its front. Making this operator explicit gives the required representation. Formally, we proceed by induction over the structure of γ, considering the following cases:

$\gamma = \delta_1 \vee \delta_2$. Then at least one of δ_1, δ_2 should have a positive weight. Suppose without loss of generality that it is δ_1. By inductive hypothesis $\delta_1 = \langle\alpha_1\rangle_I \cdot \varphi_1 \vee \beta_1$. Hence, $\gamma = \langle\alpha_1\rangle_I \cdot \varphi_1 \vee (\beta_1 \vee \delta_2)$ and we obtain the required decomposition (3) with $\alpha = \alpha_1$, $\varphi = \varphi_1$ and $\beta = \beta_1 \vee \delta_2$.

$\gamma = \delta_1 \cdot \delta_2$. If $\zeta(\delta_1) > 0$, then by inductive hypothesis $\delta_1 = \langle\alpha_1\rangle_I \cdot \varphi_1 \vee \beta_1$. Then the representation

$$\gamma = \langle\alpha_1\rangle_I \cdot (\varphi_1 \cdot \delta_2) \vee (\beta_1 \cdot \delta_2)$$

has the required form (3) with $\alpha = \alpha_1$, $\varphi = \varphi_1 \cdot \delta_2$ and $\beta = \beta_1 \cdot \delta_2$.

Otherwise, if $\zeta(\delta_1) = 0$, then, according to the definition of $\zeta(\delta_1 \cdot \delta_2)$, $\varepsilon \in \delta_1$ and $\zeta(\delta_2) = \zeta(\gamma)$ is positive. By inductive hypothesis, $\delta_2 = \langle\alpha_2\rangle_I \cdot \varphi_2 \vee \beta_2$. In this case, the required representation is

$$\gamma = (\varepsilon \vee \nu(\delta_1)) \cdot \delta_2 = \delta_2 \vee \nu(\delta_1) \cdot \delta_2 = \langle\alpha_2\rangle_I \cdot \varphi_2 \vee (\beta_2 \vee \nu(\delta_1) \cdot \delta_2)$$

$\gamma = \delta_1^*$. In this case, $\zeta(\delta_1) = \zeta(\gamma)$ is positive and by the inductive hypothesis $\delta_1 = \langle\alpha_1\rangle_I \cdot \varphi_1 \vee \beta_1$ with α_1 ε-free. We can represent γ as follows:

$$\gamma = \nu(\delta_1) \cdot \delta_1^* \vee \varepsilon = \langle\alpha_1\rangle_I \cdot (\varphi_1 \cdot \delta_1^*) \vee (\nu(\beta_1) \cdot \delta_1^* \vee \varepsilon),$$

which is in the required form.

$\gamma = \langle\delta\rangle_I$. If δ is ε-free, then γ is already in the required form with $\alpha = \delta$ and $\varphi = \varepsilon$. Otherwise, if $\varepsilon \in \delta$, then either $\gamma = \langle\nu(\delta)\rangle_I \cdot \varepsilon \vee \varepsilon$ or $\gamma = \langle\nu(\delta)\rangle_I \cdot \varepsilon \vee \emptyset$ depending on whether or not $0 \in I$.

The reader can verify that in all the cases the equality $\zeta(\gamma) = \zeta(\alpha) + \zeta(\beta) + 1$ is preserved. □

The proof of elimination of absorbing concatenation and iteration proceeds by induction on the weight of the expression. The following two lemmata establish the base case (slow expressions of weight 0) and the inductive step.

LEMMA 3.6 (ELIMINATION FOR SLOW EXPRESSIONS). *If γ is slow, then*

$$\delta \circ \gamma = \delta \cdot \gamma; \quad \delta \circ (\varepsilon \vee \gamma) = \delta \circ \varepsilon \vee \delta \circ \gamma = \langle\delta\rangle_0 \vee \delta \cdot \gamma \tag{4}$$

and

$$\gamma^{\circledast} = \gamma^*; \quad (\varepsilon \vee \gamma)^{\circledast} = \gamma^* \tag{5}$$

The inductive step is based on the following identities:

LEMMA 3.7 (ELIMINATION BY WEIGHT REDUCTION). *For any three languages α, φ, β, such that α is ε-free, and any interval I, the following equalities hold*:

$$\delta \circ (\langle\alpha\rangle_I \cdot \varphi \vee \beta) = \langle\delta \circ \alpha\rangle_I \cdot \varphi \vee \delta \circ \beta \tag{6}$$

and

$$\begin{aligned}(\langle\alpha\rangle_I \cdot \varphi \vee \beta)^{\circledast} \;=\; & \beta^{\circledast} \vee \langle\beta^{\circledast} \circ \alpha\rangle_I \cdot \varphi \circ \beta^{\circledast} \vee \\ & \langle\langle\beta^{\circledast} \circ \alpha\rangle_I \cdot \varphi \circ (\alpha \cdot \varphi \vee \beta)^{\circledast} \circ \alpha\rangle_I \cdot \varphi \circ \beta^{\circledast}.\end{aligned} \tag{7}$$

PROOF. Equation (6) follows immediately from the definition of absorbing concatenation and from Proposition 3.2. The first line of Eq. (7) corresponds to the case when $\alpha \cdot \varphi$ never occurs in the sequence, the second line—to the case when it occurs only once. The last line corresponds to the case when it occurs twice or more. For this case, it is sufficient to restrict to the interval I only the termination times of the first and the last occurrences of α. By virtue of the convexity of I, this guarantees that all other occurrences of α between them also fit in this interval. □

PROPOSITION 3.8 (ELIMINATION OF ABSORBING OPERATIONS). *Let M and L be regular timed languages, that is, defined by expressions in $\mathcal{E}$. Then*:

(1) *The language $L \circ M$ is regular.*
(2) *The language $L^{\circledast}$ is regular.*

The regular expressions for these languages can be obtained algorithmically.

PROOF. The proof of both facts is by induction on the weight, where the base case is covered by Lemma 3.6. The inductive step for $\circ$ can be made as follows. Given an expression γ of a nonzero weight, we convert it in accordance with Lemma 3.5 to the form $\gamma = \langle\alpha\rangle_I \cdot \varphi \vee \beta$ with $\zeta(\alpha), \zeta(\beta) < \zeta(\gamma)$ and $\varepsilon \notin \alpha$. Now we use the identity (6) of Lemma 3.7. The regularity of the right hand follows from the inductive hypothesis since both $\delta \circ \alpha$ and $\delta \circ \beta$ have smaller weight. This proves the first statement of Proposition 3.8.

Using this proposition and Lemma 3.7, the inductive step for $^{\circledast}$ is immediate: given an expression γ of a nonzero weight we take its representation $\gamma = \langle\alpha\rangle_I \cdot \varphi \vee \beta$. Then we apply the identity (7). Its right-hand side is regular by inductive hypothesis, since $^{\circledast}$ is applied there only to expressions of weight smaller than γ. Hence, L is regular and this concludes the proof of Proposition 3.8. Clearly, recursive algorithms for elimination of $\circ$ and $^{\circledast}$ can be derived from this proof. □

The following result is now immediate.

THEOREM 3.9. *$\mathcal{E}\mathcal{E}(\Sigma)$ has the same expressive power as $\mathcal{E}(\Sigma)$.*

As an example, let us eliminate $\circ$ from

$$\delta = \langle \underline{d} \rangle_3 \circ \left(\left\langle \langle \underline{a} \rangle_{[1,6]} \cdot \underline{b} \right\rangle_8 \cdot \underline{c} \right)^*$$

First, transform the second term to the form:

$$\left(\left\langle \langle \underline{a} \rangle_{[1,6]} \cdot \underline{b} \right\rangle_8 \cdot \underline{c} \right)^* = \left\langle \langle \underline{a} \rangle_{[1,6]} \cdot \underline{b} \right\rangle_8 \cdot \underline{c} \cdot \left(\left\langle \langle \underline{a} \rangle_{[1,6]} \cdot \underline{b} \right\rangle_8 \cdot \underline{c} \right)^* \vee \varepsilon$$

and then compute

$$\begin{aligned} \delta &= \left\langle \langle \underline{d} \rangle_3 \circ \left(\langle \underline{a} \rangle_{[1,6]} \cdot \underline{b} \right) \right\rangle_8 \cdot \underline{c} \cdot \left(\left\langle \langle \underline{a} \rangle_{[1,6]} \cdot \underline{b} \right\rangle_8 \cdot \underline{c} \right)^* \vee \langle \langle \underline{d} \rangle_3 \rangle_0 \\ &= \left\langle \langle \langle \underline{d} \rangle_3 \cdot \underline{a} \rangle_{[1,6]} \cdot \underline{b} \right\rangle_8 \cdot \underline{c} \cdot \left(\left\langle \langle \underline{a} \rangle_{[1,6]} \cdot \underline{b} \right\rangle_8 \cdot \underline{c} \right)^*. \end{aligned}$$

An example of elimination of absorbing iteration (applied to the language of a timed automaton) can be found at the end of Section 6.6.

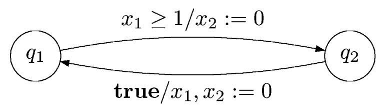

FIG. 2. A timed automaton.

4. *Timed Automata and Their Languages*

This section introduces timed automata as recognizers of timed languages, starting with an informal illustration of the structure and the behavior of timed automata. Consider the timed automaton of Figure 2. It has two states and two clocks x_1 and x_2. Suppose it starts operating in the configuration $(q_1, 0, 0)$ where the last two coordinates denote the values of the clocks. When the automaton stays at q_1, the values of the clocks grow at a uniform rate. After one second, the condition $x_1 \geq 1$ (the guard of the transition from q_1 to q_2) is satisfied and the automaton *can* move to q_2 while resetting x_2 to 0. Having entered q_2 at a configuration $(q_2, t, 0)$ for some t, the automaton can either stay there or can unconditionally move to q_1 and reset the two clocks. By fixing some initial and final states, and by assigning letters from Σ to some transitions, we can turn timed automata into generators or acceptors of timed languages, that is, sets of time-event sequences. The definition below is a minor modification of the original definition in Alur and Dill [1994].

Definition 4.1. A *Timed Automaton* is a tuple $\mathcal{A} = (Q, C, \Delta, \Sigma, s, F)$ where Q is a finite set of states, C is a finite set of clocks, Σ is an input (or event) alphabet, Δ is a transition relation (see below), $s \in Q$ an initial state and $F \subset Q$ a set of accepting states. The transition relation consists of tuples of the form (q, ϕ, ρ, a, q') where q and q' are states, $a \in \Sigma \cup \{\varepsilon\}$ is a letter, $\rho \subseteq C$ and ϕ (the transition guard) is a Boolean combination of formulas of the form $(x \in I)$ for some clock x and some integer-bounded interval I.

A *clock valuation* is a function $\mathbf{v} : C \to \mathbb{R}_+$, or equivalently a $|C|$-dimensional vector over $\mathbb{R}_+$. We denote the set of all clock valuations by $\mathcal{H}$. A configuration of the automaton is hence a pair $(q, \mathbf{v}) \in Q \times \mathcal{H}$ consisting of a discrete state (sometimes called "location") and a clock valuation. Every subset $\rho \subseteq C$ induces a reset function $\text{Reset}_\rho : \mathcal{H} \to \mathcal{H}$ defined for every clock valuation $\mathbf{v}$ and every clock variable $x \in C$ as

$$\text{Reset}_\rho\, \mathbf{v}(x) = \begin{cases} 0 & \text{if } x \in \rho \\ \mathbf{v}(x) & \text{if } x \notin \rho. \end{cases}$$

That is, Reset_ρ resets to zero all the clocks in ρ and leaves the other clocks unchanged. We use $\mathbf{1}$ to denote the unit vector $(1, \ldots, 1)$ and $\mathbf{0}$ for the zero vector.

Definition 4.2 (*Steps, Runs and Acceptance*). A step of the automaton is one of the following:

—A discrete step:

$$(q, \mathbf{v}) \xrightarrow{a} (q', \mathbf{v}'),$$

where $a \in \Sigma \cup \{\varepsilon\}$ and there exists $\delta = (q, \phi, \rho, a, q') \in \Delta$, such that $\mathbf{v}$ satisfies ϕ and $\mathbf{v}' = \text{Reset}_\rho(\mathbf{v})$.

—A time step:

$$(q, \mathbf{v}) \xrightarrow{t} (q, \mathbf{v} + t\mathbf{1}),$$

where $t \in \mathbb{R}_+$.

A finite run of a timed automaton is a finite sequence of steps

$$(q_0, \mathbf{v}_0) \xrightarrow{z_1} (q_1, \mathbf{v}_1) \xrightarrow{z_2} \cdots \xrightarrow{z_n} (q_n, \mathbf{v}_n).$$

The *trace* of a run is the time-event sequence $z_1 \cdot z_2 \cdots z_n$. A *trivial* run is just a configuration $(q, \mathbf{v})$, and its trace is ε.

An *accepting* run is a run starting from the initial configuration $(s, \mathbf{0})$ and terminating by a discrete step to a final state, that is, $q_n \subset F$ and z_n is a discrete step.

The language of a timed automaton, $L(\mathcal{A})$, consists of all the traces of its accepting runs.

A slight modification of this definition is needed in order to accept signals (or signal-event sequences), namely to associate an element of the signal alphabet to each *state* of the automaton [Asarin et al. 1997]. Note also that we insist on a single initial configuration, because otherwise we can have a non-countable number of initial states and the language equations developed in Section 6 should be parameterized by clock values, resulting in a much more complicated construction.

5. *From Expressions to Timed Automata*

Here we prove the easy part of the timed version of Kleene Theorem, namely, every timed regular language can be recognized by a timed automaton. Similarly to the untimed construction in McNaughton and Yamada [1960], automata are built from expressions by induction on the structure of the expression. We make this construction in the most general settings, namely, for the class $\mathcal{GEE}$, and show that an accepting timed automaton can be built for every language defined by a (generalized extended) timed regular expression.

Before giving the formal definition let us explain the construction intuitively (see also Figure 3). The automaton for $\underline{a}$ can make, at any nonnegative time, an a-transition from the initial state to the final state. For the union of two languages, we choose nondeterministically between the two automata. To concatenate two languages, we add transitions to the initial state of the second automaton for every accepting transition of the first automaton. For standard concatenation, such transitions reset the clocks, while for absorbing concatenation the clocks are not reset. Likewise for the * operations, we add transitions to the initial state and reset all the clocks.

The construction of the automaton for $\varphi^{\circledast}$ is better understood using an extension of timed automata where upon a transition a clock can be assigned the value of another clock. The basic idea is that for every new iteration of φ we need the values of all clocks to represent the total time elapsed in the previous iterations. We achieve this by adding a new clock x that is never reset to zero and transitions to the initial state in which all clocks get the value of x (see Figure 3). Our construction below "simulates" these automata using ordinary timed automata that keep track of the clocks that have been reset. References in the guards to those clocks which have not been reset are replaced by references to x. For the $\langle\varphi\rangle_I$ operator we introduce a

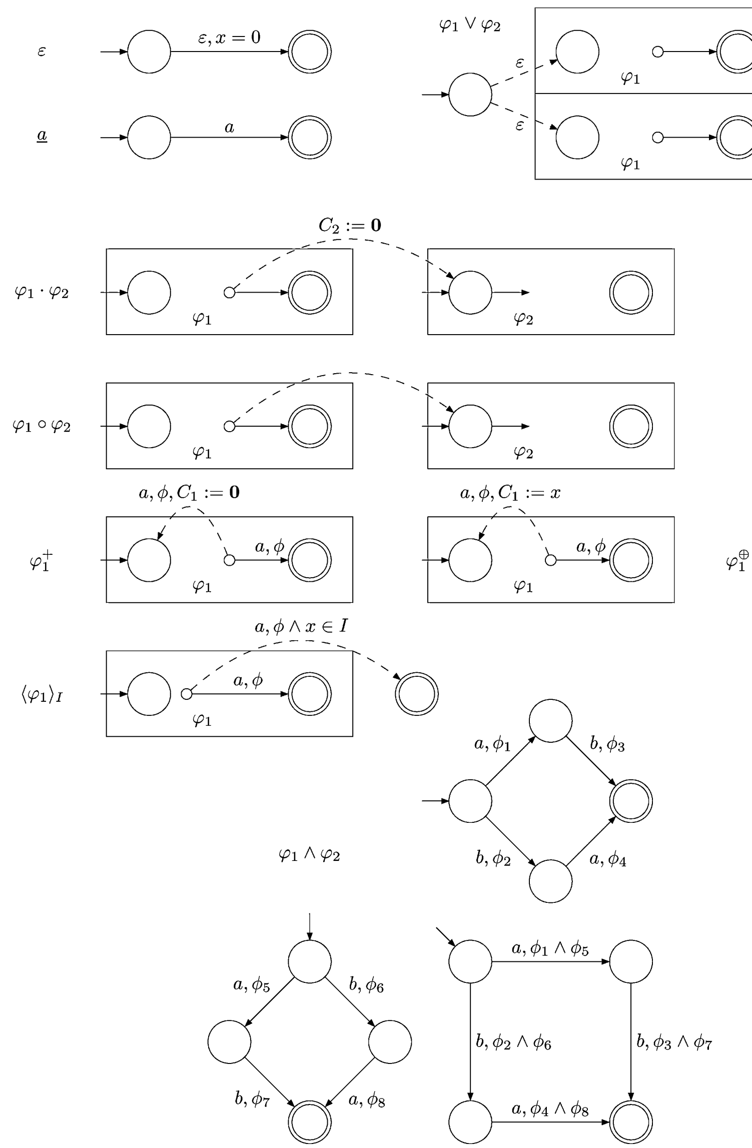

FIG. 3. Constructing automata from expressions.

new clock x and add a test $(x \in I)$ to the guard of every transition leading to f. For intersection, we do the usual Cartesian product (taking special care of ε-transitions). Finally, for renaming, we just rename the transition labels.

Definition 5.1 (*Automata from Expressions*). Let $\mathcal{A}_1 = (Q_1, C_1, \Delta_1, \Sigma, s_1, F_1)$ and $\mathcal{A}_2 = (Q_2, C_2, \Delta_2, \Sigma, s_2, F_2)$ be the timed automata accepting the languages $[\![\varphi_1]\!]$ and $[\![\varphi_2]\!]$ respectively. We assume that Q_1 and Q_2 as well as C_1 and C_2 are disjoint.

—The automaton for $[\![\varepsilon]\!]$ is $(\{s, f\}, \{x\}, \Delta, \Sigma, s, \{f\})$, where the transition relation is $\Delta = \{(s, x = 0, \emptyset, \varepsilon, f)\}$.

— The automaton for $[\![\underline{a}]\!]$, $a \in \Sigma$ is $(\{s, f\}, \emptyset, \Delta, \Sigma, s, \{f\})$, where the transition relation is $\Delta = \{(s, \mathbf{true}, \emptyset, a, f)\}$.

— The automaton for $[\![\varphi_1 \vee \varphi_2]\!]$ is $(Q_1 \cup Q_2 \cup \{s\}, C_1 \cup C_2, \Delta, \Sigma, s, F_1 \cup F_2)$, where Δ is constructed by adding to $\Delta_1 \cup \Delta_2$ two new ε-transitions $(s, x = 0, \emptyset, \varepsilon, s_i)$, where x is any clock and $i \in \{1, 2\}$ (if there is no clock in the automata we should add one).

— The automaton for $[\![\varphi_1 \wedge \varphi_2]\!]$ is $(Q_1 \times Q_2 \cup \{f\}, C_1 \cup C_2, \Delta, \Sigma, \langle s_1, s_2\rangle, \{f\})$, where Δ contains

  — a transition $\{(\langle q_1, q_2\rangle, \phi_1 \wedge \phi_2, \rho_1 \cup \rho_2, a, \langle q_1', q_2'\rangle)$ for any $(q_1, \phi_1, \rho_1, a, q_1') \in \Delta_1$ and any $(q_2, \phi_2, \rho_2, a, q_2') \in \Delta_2\}$;

  — a transition $\{(\langle q_1, q_2\rangle, \phi_1 \wedge \phi_2, \rho_1 \cup \rho_2, a, f)$ for any $(q_1, \phi_1, \rho_1, a, f_1) \in \Delta_1$ and any $(q_2, \phi_2, \rho_2, a, f_2) \in \Delta_2\}$ where $f_1 \in F_1$ and $f_2 \in F_2$;

  — a transition $\{(\langle q_1, q_2\rangle, \phi_1, \rho_1, \varepsilon, \langle q_1', q_2\rangle)$ for any $(q_1, \phi_1, \rho_1, \varepsilon, q_1') \in \Delta_1\}$;

  — a transition $\{(\langle q_1, q_2\rangle, \phi_2, \rho_2, \varepsilon, \langle q_1, q_2'\rangle)$ for any $(q_2, \phi_2, \rho_2, \varepsilon, q_2') \in \Delta_2\}$

— The automaton for $[\![\varphi_1 \cdot \varphi_2]\!]$ is $(Q_1 \cup Q_2, C_1 \cup C_2, \Delta, \Sigma, s_1, F_2)$ where Δ is constructed from $\Delta_1 \cup \Delta_2$ by inserting for every transition of the form $(q_1, \phi, \rho, a, f_1)$ in Δ_1 with $f_1 \in F_1$ a new transition (q_1, ϕ, C_2, a, s_2). The automaton for $[\![\varphi_1 \circ \varphi_2]\!]$ is the same except for the fact that the new transition is of the form $(q_1, \phi, \emptyset, a, s_2)$.

— The automaton for $[\![\varphi_1^+]\!]$ is $\mathcal{A} = (Q_1, C_1, \Sigma, \Delta, s_1, F_1)$ where Δ is constructed from Δ_1 by adding for every transition of the form (q, ϕ, ρ, a, f_1) in Δ_1 with $f_1 \in F_1$ a transition of the form (q, ϕ, C_1, a, s_1).

— The automaton for $[\![\varphi_1^\oplus]\!]$ is $\mathcal{A} = (Q_1 \times 2^{C_1}, C_1 \cup \{x\}, \Sigma, \Delta, (s_1, \emptyset), F_1 \times 2^{C_1})$. The second component of the state records which clocks have been reset during the current iteration of $[\![\varphi_1]\!]$. There are two types of transitions in Δ:

  — *Transitions Simulating Those of $\mathcal{A}_1$*. For every transition of the form (q, ϕ, ρ, a, q') in Δ_1 and every $D \subset C_1$ the relation Δ contains $((q, D), \phi_D, \rho, a, (q', D \cup \rho))$;

  — *Looping Transitions*. For every transition of the form $(q_1, \phi, \rho, a, f_1)$ in Δ_1 with $f_1 \in F_1$ and every $D \subset C_1$, the relation Δ contains $((q_1, D), \phi_D, \rho, a, (s_1, \emptyset))$.

  Here ϕ_D is obtained by replacing in ϕ all occurrences of clocks not belonging to D by x.

— The automaton for $[\![\varphi_1^*]\!]$ (respectively, $[\![\varphi_1^\circledast]\!]$) is obtained by the union construction from the automaton for $\{\varepsilon\}$ and the automaton for $[\![\varphi_1^+]\!]$ (respectively, for $[\![\varphi_1^\oplus]\!]$).

— The automaton for $[\![\langle\varphi_1\rangle_I]\!]$ is $\mathcal{A} = (Q_1 \cup \{f\}, C_1 \cup \{x\}, \Delta, \Sigma, s_1, \{f\})$ where Δ is obtained from Δ_1 by introducing for every transition of the form (q, ϕ, ρ, a, f_1) in Δ_1 with $f_1 \in F_1$ a new transition $(q, \phi \wedge (x \in I), \rho, a, f)$.

— The automaton for $[\![\theta(\varphi_1)]\!]$ and $\theta : \Sigma \to \Sigma'$ is $\mathcal{A} = (Q_1, C_1, \Delta, \Sigma', s_1, F_1)$ where Δ is obtained from Δ_1 by replacing every transition of the form (q, ϕ, ρ, a, q') in Δ_1 by $(q, \phi, \rho, \theta(a), q')$.

This concludes the construction that gives one side of Kleene theorem:

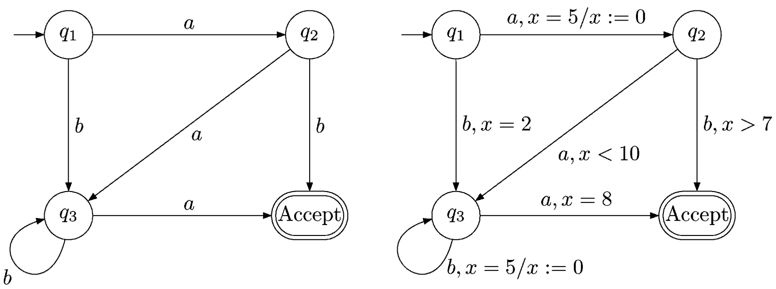

FIG. 4. A timed and an untimed automaton.

THEOREM 5.2 (EXPRESSIONS $\Rightarrow$ AUTOMATA). *Every timed language defined by a (generalized extended) regular expression is accepted by a timed automaton.*

6. *From Timed Automata to Expressions*

6.1. THE APPROACH. Our proof of the other (and harder) side of Kleene theorem is modeled after the proof of the classical theorem given in McNaughton and Yamada [1960], which constructs from an automaton a system of linear language equations of the form:

$$X_i = \alpha_i \vee \bigvee_{j=1}^{n} \beta_{ij} \cdot X_j \qquad i = 1, \ldots, n, \tag{8}$$

where the X_i stand for unknown languages and α_i, β_{ij}—for given regular coefficients. Each unknown X_i of the system corresponds to the language accepted by the automaton starting from state q_i. As an example consider the first (untimed) automaton on Figure 4. The languages associated with its states satisfy the following self-explanatory system of equations:

$$\begin{aligned} X_3 &= a \vee b \cdot X_3 \\ X_2 &= b \vee a \cdot X_3 \\ X_1 &= a \cdot X_2 \vee b \cdot X_3. \end{aligned} \tag{9}$$

Using the well-known fact [Arden 1960] that any equation of the form

$$X = \alpha \vee \beta \cdot X$$

admits a minimal solution

$$X = \beta^* \cdot \alpha,$$

it can be proved that any system of equations such as (8) has a regular minimal solution and a corresponding regular expression can be found effectively from the coefficients. If, in addition, $\varepsilon \notin \beta_{ij}$ then the solution is unique. For example, the solution for (9) is:

$$\begin{aligned} X_3 &= b^* \cdot a \\ X_2 &= b \vee a \cdot b^* \cdot a \\ X_1 &= a \cdot (b \vee a \cdot b^* \cdot a) \vee b^+ \cdot a. \end{aligned}$$

Adapting this proof to timed automata is problematic as the timed automaton of Figure 4 shows. In this automaton, the transition from q_1 to q_2 resets the clock and hence a fragment of the equation for q_1 will be $X_1 = \langle \underline{a} \rangle_5 \cdot X_2 \vee \cdots$; however, we cannot do the same and use $\langle \underline{b} \rangle_2 \cdot X_3$ for that part of X_1 accepted via q_3, because after completing action b the automaton enters state q_3 with a clock value other than zero. To tackle this problem we could associate a language with every configuration of the timed automaton, that is, let $X_{i,v}$ denote the language accepted starting from state q_i and clock valuation v. This would lead to an infinite number of variables and equations. We use an alternative solution, namely associate X_i with the language accepted from $(q_i, 0)$ and use the absorbing concatenation for non-resetting transitions. The system of equations for the automaton is thus

$$\begin{aligned} X_3 &= \langle \underline{a} \rangle_8 \vee \langle \underline{b} \rangle_5 \cdot X_3 \\ X_2 &= \langle \underline{b} \rangle_{(7,\infty)} \vee \langle \underline{a} \rangle_{[0,10)} \circ X_3 \\ X_1 &= \langle \underline{a} \rangle_5 \cdot X_2 \vee \langle \underline{b} \rangle_2 \circ X_3. \end{aligned}$$

Such "quasi-linear" equations, which use both kinds of concatenation, can be written for any *one-clock* automaton. However, when an automaton $\mathcal{A}$ has several clocks, the set of transitions cannot be partitioned into resetting and nonresetting ones, and we need first to split the automaton into several one-clock automata, the intersection of their languages gives the language of $\mathcal{A}$. For each such automaton, we define the corresponding equations and by showing how such equations can be solved the proof of Kleene theorem will be completed.

6.2. FROM TIMED AUTOMATA TO ONE-CLOCK AUTOMATA. The reduction into one-clock automata starts with a language-preserving transformation on the automaton, which eliminates undesirable features as a preparation for the translation into expressions. Then we "determinize" the automaton by assigning a distinct letter to every transition outgoing from any state. Having done that we can split the automaton into several one-clock automata from which language equations are constructed.

An automaton is *disjunction-free* if for every transition (q, ϕ, ρ, a, q'), the formula ϕ is a conjunction of simple tests $(x \in I)$ and their negations. An automaton is *strongly deterministic* if it contains no ε-transitions and for any state q and any letter a the transition relation contains at most one outgoing transition from q labeled by a. Note that strong determinism is a syntactic property which is sufficient but not necessary for determinism—the latter can be implied by empty intersections of guards for two transitions labeled by the same letter.

LEMMA 6.1 (DISJUNCTION-FREE AND STRONGLY-DETERMINISTIC AUTOMATA). *From any timed automaton $\mathcal{A}$ over Σ one can construct a disjunction-free and strongly deterministic automaton $\mathcal{A}'$ over Σ', and a renaming $\theta : \Sigma \to \Sigma'$ such that $L(\mathcal{A}) = \theta(L(\mathcal{A}'))$.*

PROOF. To get rid of disjunctions we first convert every transition guard into a disjunctive normal form (DNF) $\phi = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_k$ where every ϕ_i is a conjunction. We then replace every transition $\delta = (q, \phi, \rho, a, q')$, where $\phi = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_k$ by k transitions of the form (q, ϕ_i, ρ, a, q'), $i = 1, \ldots, k$. Clearly, this automaton accepts $L(\mathcal{A})$. Any disjunction-free automaton $\mathcal{A} = (Q, C, \Delta, \Sigma, s, F)$ can be converted into a strongly deterministic automaton $\mathcal{A}' = (Q, C, \Delta', \Sigma \times \{1..M\}, s, F)$, where M is the maximal number of transitions with the same label outgoing from the same state, Δ' is obtained from Δ by replacing any transition (q, ϕ, ρ, a, q') by $(q, \phi, \rho, (a, i), q')$, choosing a different i component for each transition a going from state q. For the renaming $\theta : (\Sigma \cup \{\varepsilon\}) \times \{1..M\} \to \Sigma \cup \{\varepsilon\}$ defined by the formula $\theta(a, i) = a$ we have the language equality $\theta(L(\mathcal{A}')) = L(A)$. □

THEOREM 6.2 (REDUCTION TO ONE-CLOCK AUTOMATA). *Let $\mathcal{A}$ be a timed automaton with k clocks. One can build k one-clock automata $\mathcal{A}_1, \ldots, \mathcal{A}_k$ and a renaming θ such that*

$$L(\mathcal{A}) = \theta\left(\bigcap_{i=1}^{k} L(\mathcal{A}_i)\right).$$

PROOF. First, we transform $\mathcal{A}$ into a disjunction-free and strongly deterministic form $\mathcal{A}' = (Q', C, \Delta', \Sigma', s', F')$ and find a renaming θ such that $L(\mathcal{A}) = \theta(L(\mathcal{A}'))$. Let $C = \{x_1, \ldots, x_k\}$. We separate $\mathcal{A}'$ into k automata $\mathcal{A}_i = (Q', \{x_i\}, \Delta'_i, \Sigma', s', F')$ such that for every $(q, \phi, \rho, a, q') \in \Delta'$ there is $(q, \phi_i, \rho_i, a, q') \in \Delta_i$ such that $\rho_i = \rho \cap \{x_i\}$ and ϕ_i is obtained from ϕ by substituting **true** in every occurrence of $x_j \in I$ or of $x_j \notin I$ for every $j \neq i$. In other words, every $\mathcal{A}_i$ respects only the constraints imposed by the clock x_i and ignores the rest of the clocks. Since the automaton $\mathcal{A}'$ is strongly-deterministic, every accepted sequence is a trace of exactly *one* run, and this is *the same* run in every $\mathcal{A}_i$. A run is possible in every $\mathcal{A}_i$ iff it is possible in $\mathcal{A}'$. □

An example of the translation appears in Section 6.6.

6.3. EQUATIONS FOR TIMED AUTOMATA. From one-clock automata, we derive timed language equations involving the $\circ$ operation and whose solutions involve also the $^{\circledast}$ operation. Both can later be eliminated using the procedure described in Section 3.

Definition 6.3 (*Quasilinear Equations*). A system of quasilinear timed language equations has the following form:

$$X_i = \alpha_i \vee \bigvee_{j=1}^{n} \beta_{ij} \cdot X_j \vee \bigvee_{j=1}^{n} \gamma_{ij} \circ X_j, \qquad i = 1, \ldots, n, \tag{10}$$

where the X_i stands for unknown timed languages and the coefficients $\alpha_i, \beta_{ij}, \gamma_{ij}$— for given timed languages.

To avoid some complication with non-unique solutions (and non-associative multiplication), we consider only *normal systems of equations* where all coefficients satisfy

$$\beta_{ij} \cap \mathbb{R}_+ = \emptyset; \;\; \gamma_{ij} \cap \mathbb{R}_+ = \emptyset; \tag{11}$$

that is, any sequence in any coefficient language except the α_i's should contain at least one discrete event from Σ.

Definition 6.4 (*From One-Clock Automata to Equations*). Let $\mathcal{A} = (Q, \{x\}, \Delta, \Sigma, s, F)$ be a one-clock automaton. The system of equations associated with $\mathcal{A}$ is (10) with an unknown X_i for every $q_i \in Q$. The coefficient α_i is the disjunction of expressions $\langle \underline{a} \rangle_I$ for all the transitions $(q_i, x \in I, \rho, a, f) \in \Delta$ with $f \in F$. The coefficients β_{ij}, γ_{ij} are constructed from the transitions in Δ as follows:

Transition	Coefficient
$(q_i, x \in I, \{x\}, a, q_j)$	$\beta_{ij} = \langle \underline{a} \rangle_I$
$(q_i, x \in I, \emptyset, a, q_j)$	$\gamma_{ij} = \langle \underline{a} \rangle_I$

Note that if the transition guard of the transition is **true**, then the corresponding coefficient is just $\underline{a}$.

The following self-evident lemma specifies the connection between the language of a timed automaton and the constructed equations.

LEMMA 6.5. *Let L_i be the language accepted by the automaton from configuration* $(q_i, 0)$. *Then $X_1 = L_1, \ldots, X_n = L_n$ is a solution of Eq.* (10).

6.4. SOLVING QUASILINEAR EQUATIONS. The rest of this section is devoted to the description of the solution algorithm, which is an adaptation of the standard Gaussian elimination procedure used for linear equations.

The following lemma gives a solution to a single equation with only one operation.

LEMMA 6.6. *Let α, β, γ be timed languages.*

(1) *The smallest solution to $X = \alpha \vee \gamma \circ X$ is $X^0 = \gamma^{\circledast} \circ \alpha$;*

(2) *The smallest solution to $Y = \alpha \vee \beta \cdot Y$ is $Y^0 = \beta^* \cdot \alpha$;*

(3) *If β and γ satisfy the normality condition* (11) *then these solutions are unique.*

PROOF. The proof is similar to the proof of the same result for untimed equations; we give a sketch only for the absorbing concatenation.

First, we verify that X^0 is a solution by substituting it into the right-hand side of the equation:

$$\alpha \vee \gamma \circ X^0 = \alpha \vee \gamma \circ \gamma^{\circledast} \circ \alpha = (\varepsilon \vee \gamma^{\oplus}) \circ \alpha = X^0.$$

The minimality proof proceeds as follows: Let X^1 be a solution, that is, $X^1 = \alpha \vee \gamma \circ X^1$. The inclusion $X^0 = \gamma^{\circledast} \circ \alpha \subset X^1$ follows from the following statement, which can be proved by straightforward induction over n:

$$\forall n \; (\gamma^{\circ n} \circ \alpha \subset X^1).$$

In order to prove uniqueness (under normality hypothesis) we introduce the *discrete length* η of time-event sequences. The morphism $\eta : \mathcal{T} \to \mathbb{N}$ is defined by $\eta(r) = 0$ for all $r \in \mathbb{R}_+$ and $\eta(a) = 1$ for all $a \in \Sigma$. Note that $\eta(u \circ v) = \eta(u) + \eta(v)$ whenever $u \circ v$ is defined.

The proof of the inclusion $X^1 \subset X^0$ uses the normality condition on γ and proceeds by contradiction. Suppose the inclusion does not hold and let w be a sequence in $X^1 - X^0$ with the minimal possible discrete length $\eta(w)$. Since X^1 is a solution, $w \in \alpha \vee \gamma \circ X^1$. The sequence w cannot belong to $\alpha \subset X^0$. Hence, w admits a decomposition $v = u \circ v$ with $u \in \gamma$ and $v \in X^1$. The normality condition guarantees that $\eta(u) > 0$, hence $\eta(v) = \eta(w) - \eta(u) < \eta(w)$. Since $\eta(w)$ is minimal in $X^1 - X^0$, this implies that $v \in X^0$; hence,

$$w = u \circ v \in \gamma \circ X^0 = \gamma \circ \gamma^{\circledast} \circ \alpha = \gamma^{\oplus} \circ \alpha \subseteq X^0,$$

which contradicts the hypothesis on w and concludes the proof. $\square$

In the sequel, we use this lemma in a specific situation when α can depend on X. To justify such a usage we prove the following statement.

COROLLARY 6.7. *Suppose that β and γ satisfy the normality condition* (11), *and $h(X)$ is any language-valued expression depending on X. Then*

—*the equation $X = h(X) \vee \gamma \circ X$ is equivalent to $X = \gamma^{\circledast} \circ h(X)$;*
—*the equation $Y = h(X) \vee \beta \cdot Y$ is equivalent to $Y = \beta^{*} \cdot h(X)$.*

PROOF. The proofs of the two statements are similar and we give only the first one. Let X^0 be a timed language. It is a solution of the first equation whenever it satisfies $X^0 = h(X^0) \vee \gamma \circ X^0$, or, equivalently, whenever it is a solution of the equation $X = h(X^0) \vee \gamma \circ X$. The language X^0 is a solution of this equation if and only if it is equal to its unique solution provided by Lemma 6.6, that is, $X^0 = \gamma^{\circledast} \circ h(X^0)$. The last equality holds if and only if X^0 is a solution to the equation $X = \gamma^{\circledast} \circ h(X)$. This concludes the proof of equivalence of the two equations. □

THEOREM 6.8. *A normal system of quasilinear equations has one and only one solution. This solution is regular. Its regular expression can be obtained algorithmically from expressions for the coefficients.*

PROOF. The algorithm for solving the system (10) consists in iterated applications of Corollary 6.7. It has four stages, the first two treat the $\circ$ operation and the next two—the standard concatenation.

At the first stage, we use the first equation and Corollary 6.7 to express X_1 as

$$X_1 = \gamma_{11}^{\circledast} \circ \left(\alpha_1 \vee \bigvee_{j=1}^{n} \beta_{1j} \cdot X_j \vee \bigvee_{j=2}^{n} \gamma_{1j} \circ X_j \right).$$

Notice that only the occurrence of $\circ X_1$ is eliminated, while those of $\cdot X_1$ remain in the equation. By opening the parentheses (using Proposition 3.2, whose assumptions are satisfied because the system is normal), this equation is transformed to the form

$$X_1 = \alpha_1' \vee \bigvee_{j=1}^{n} \beta_{1j}' \cdot X_j \vee \bigvee_{j=2}^{n} \gamma_{1j}' \circ X_j.$$

We substitute this expression into the $\circ X_1$ occurrence of X_1 in the second equation, solve it for X_2 and so on until X_n for which we find an expression that contains only occurrences of unknowns of the form $\cdot X$ and not $\circ X$. Then the second stage starts by going backwards, putting the expression for X_n into equation number $n-1$. This allows to find for X_{n-1} an expression free from occurrences of $\circ X_n$, until we reach X_1 once again. Now the system has a standard $\circ$-free form

$$X_i = \alpha_i'' \vee \bigvee_{j=1}^{n} \beta_{ij}'' \cdot X_j, \tag{12}$$

which is the starting point of the standard solution procedure for equations over Kleene algebra. We repeat the same procedure by expressing X_1 as

$$X_1 = \beta_{11}^{''*} \cdot \left(\alpha_1'' \vee \bigvee_{j=2}^{n} \beta_{1j}'' \cdot X_j \right),$$

put the result into the second equation, find X_2 and so on. The fourth (and last) stage consists in going backwards putting the expression for X_n into equation $n-1$ and so on. This ends up with finding an extended regular expression for every X_i and concludes the algorithm and the proof of Theorem 6.8. $\square$

COROLLARY 6.9. *From a one-clock automaton, one can construct an extended timed regular expression that denotes its language.*

6.5. MAIN RESULTS. Since $\mathcal{EE}$ are equivalent to $\mathcal{E}$ (and hence languages defined by extended timed regular expression are regular), Corollary 6.9 concludes the new proof of the following important result.

THEOREM 6.10. *The language accepted by any one-clock automaton is regular.*

Together with the reduction of Theorem 6.2, this gives:

THEOREM 6.11 (AUTOMATA $\Rightarrow$ EXPRESSIONS). *Every language accepted by a timed automaton can be represented by the expression*

$$\theta\left(\bigwedge_{i=1}^{k}\varphi_i\right),$$

where θ is a renaming and φ_i are timed regular expressions.

We have proved the main result of this paper:

THEOREM 6.12 (KLEENE THEOREM FOR TIMED AUTOMATA). *Timed automata and generalized timed regular expressions have the same expressive power.*

6.6. FROM AUTOMATA TO EXPRESSIONS: AN EXAMPLE. Consider the automaton $\mathcal{A}$ in Figure 5. Getting rid of disjunctions, we obtain $\mathcal{A}'$. By splitting a into d and e, and labeling the ε-transition by c, we get the strongly deterministic automaton $\mathcal{A}''$ which is separated into two one-clock automata $\mathcal{A}_1$ and $\mathcal{A}_2$. Hence,

$$L(\mathcal{A}) = L(\mathcal{A}') = \theta(L(\mathcal{A}'')) = \theta(L(\mathcal{A}_1)\cap L(\mathcal{A}_2)). \tag{13}$$

To find the expression for $L(\mathcal{A}_1)$, we write the language equations

$$\begin{aligned} U &= \left(\langle\underline{d}\rangle_{[3,\infty)} \vee \underline{e}\right)\circ V \vee c\circ W \vee c\\ V &= \underline{b}\cdot U\\ W &= \emptyset. \end{aligned}$$

After substituting $\underline{b}\cdot U$ instead of V and $\emptyset$ instead of W, we obtain:

$$U = \left(\left(\langle\underline{d}\rangle_{[3,\infty)} \vee \underline{e}\right)\circ \underline{b}\right)\cdot U \vee c,$$

which can be immediately solved using Lemma 6.6:

$$L(\mathcal{A}_1) = U = \left(\left(\langle\underline{d}\rangle_{[3,\infty)} \vee \underline{e}\right)\circ \underline{b}\right)^*\cdot c.$$

For $\mathcal{A}_2$, the equations are

$$\begin{aligned} X &= \left(\underline{d} \vee \langle\underline{e}\rangle_{(1,9)}\right)\circ Y \vee \underline{c}\circ Z \vee \underline{c}\\ Y &= \underline{b}\circ X\\ Z &= \emptyset \end{aligned}$$

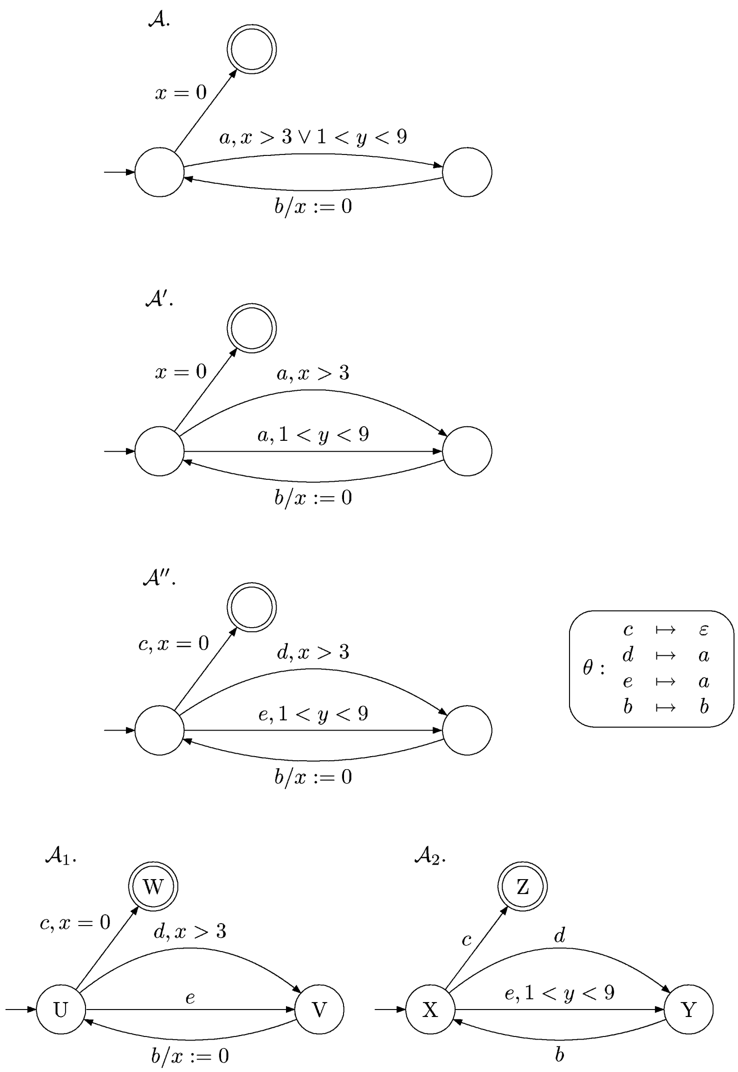

FIG. 5. Constructing an expression from an automaton.

and after substitution we get

$$X = \big(\big(\underline{d} \vee \langle \underline{e} \rangle_{(1,9)}\big) \circ \underline{b}\big) \circ X \vee \underline{c},$$

whose solution is

$$L(\mathcal{A}_2) = X = \big(\big(\underline{d} \vee \langle \underline{e} \rangle_{(1,9)}\big) \circ \underline{b}\big)^{\circledast} \circ \underline{c}.$$

Together with Eq. (13), it gives a $\mathcal{GEE}$-class expression for $L(\mathcal{A})$:

$$L(\mathcal{A}) = \theta\Big(\big(\big(\big(\langle \underline{d} \rangle_{[3,\infty)} \vee \underline{e}\big) \circ \underline{b}\big)^{*} \cdot c\big) \wedge \big(\big(\big(\underline{d} \vee \langle \underline{e} \rangle_{(1,9)}\big) \circ \underline{b}\big)^{\circledast} \circ \underline{c}\big)\Big).$$

If we want to avoid $\circ$ and $^{\circledast}$ operations, elimination algorithms from Section 3 should be applied. It is easy for the first language:

$$L(\mathcal{A}_1) = \big(\big(\langle \underline{d} \rangle_{[3,\infty)} \vee \underline{e}\big) \cdot \underline{b}\big)^{*} \cdot c$$

but less so for the second:

$$\begin{aligned}
L(\mathcal{A}_2) &= \left(\underline{db} \vee \langle \underline{e} \rangle_{(1,9)} \underline{b}\right)^{\circledast} \underline{c} \\
&= (\underline{db})^{\circledast} \underline{c} \vee \\
&\quad \langle (\underline{db})^{\circledast} \circ \underline{e} \rangle_{(1,9)} \underline{b}) \circ (\underline{db})^{\circledast} \underline{c} \vee \\
&\quad \left\langle \langle (\underline{db})^{\circledast} \circ \underline{e} \rangle_{(1,9)} \underline{b}) \circ (\underline{db} \vee \underline{eb})^{\circledast} \circ \underline{e} \right\rangle_{(1,9)} \underline{b} \circ (\underline{db})^{\circledast} \underline{c} \\
&= (\underline{db})^{*} \underline{c} \vee \\
&\quad \langle (\underline{db})^{*} \underline{e} \rangle_{(1,9)} \underline{b})(\underline{db})^{*} \underline{c} \vee \\
&\quad \left\langle \langle (\underline{db})^{*} \underline{e} \rangle_{(1,9)} \underline{b})(\underline{db} \vee \underline{eb})^{*} \underline{e} \right\rangle_{(1,9)} \underline{b}(\underline{db})^{*} \underline{c}.
\end{aligned}$$

7. *Infinitary Timed Languages*

7.1. INFINITE SEQUENCES, ω-LANGUAGES AND ω-AUTOMATA. For untimed sequences and automata, the theory of ω-languages (languages whose elements are infinite sequences) is not as nicely algebraic as the theory of finitary languages. The situation is aggravated when we move to time-event sequences where we have two notions of infinitude, metric and logical, which do not coincide.

In the finitary case an element of $\mathcal{T}(\Sigma)$ can be viewed as an alternating finite sequence $u_1 \cdot u_2 \cdots u_n$ of elements in $\mathbb{R}_+ \cup \Sigma^*$. The logical length of such a sequence is the sum of finitely many integers and its metric length is a sum of finitely many real numbers. One possibility to move to infinitary language is to define an ω-time-event sequence over Σ as an infinite alternating sequence $u_1 \cdot u_2 \cdots$ of elements from $\mathbb{R}_+ \cup \Sigma^*$. Ideally, we would like both logical and metric length to be infinite but this is not easy to guarantee in a simple way.

Concerning logical length, note that already in the untimed case, if a language L contains ε, then L^ω, the language consisting of all infinite concatenations of elements from L, might contain finite strings. Moreover, an infinite sequence might become finite under a length-reducing renaming that maps some letters to ε. Similarly, the image of an infinite time-event sequence such as

$$1 \cdot a \cdot (1 \cdot b)^\omega = 1 \cdot a \cdot 1 \cdot b \cdot 1 \cdot b \cdot 1 \cdot b \cdots$$

under a renaming which maps b to ε is the logically-finite time-event sequence $1 \cdot a \cdot \infty$. So to keep our languages closed under renaming, and to account for runs of timed automata with infinitely many ε-transitions, we allow time-event sequences with infinite metric length but with finitely many events.

Infinite metric length cannot be guaranteed locally due to the existence of converging sequences of reals. For example, the infamous infinite sequence

$$a \cdot 1 \cdot a \cdot \frac{1}{2} \cdot a \cdot \frac{1}{4} \cdots$$

due to *Zeno* of Elea has a finite metric length. Consequently, if L is a language in which there is no positive lower-bound on the metric length of its elements, for example, $L = \langle \underline{a} \rangle_{(0,r]}$, the set L^ω contains Zeno behaviors. Our design choice is to exclude explicitly such Zeno behaviors from the languages that we consider.

Definition 7.1 (*ω-Time-Event Sequences and Timed ω-Languages*). An ω-time-event sequence is an alternating (finite or infinite) sequence

$$\xi = u_1 \cdot u_2 \cdots$$

of elements in $\mathbb{R}_+ - \{0\} \cup \Sigma^+$, such that $\lambda(\xi)$ (the sum of the real elements) is infinite. When the sequence is finite, the last element must be ∞. The set of all such sequences is denoted by $\mathcal{T}_\omega(\Sigma)$ and its subsets are called (timed) ω-languages.

The concatenation $v \cdot \xi$ where $v \in \mathcal{T}(\Sigma)$ and $\xi \in \mathcal{T}_\omega(\Sigma)$ is defined almost as before, resulting in an ω-time-event sequence. For an infinite sequence $v_1, v_2, \ldots$ of time-event sequences such that $\sum_{i=1}^{\infty} \lambda(v_i) = \infty$, their infinite concatenation $\cdot_{i=1}^{\infty}$ is defined in the natural way. When extending this definition to ω-languages, by letting

$$\mathop{\cdot}_{i=1}^{\infty} L_i = \left\{ \mathop{\cdot}_{i=1}^{\infty} v_i : v_i \in L_i \right\}$$

we do not allow an arbitrary choice of v_i's but only those, whose *sum of lengths diverges*.

Definition 7.2 (*Timed ω-Regular Expressions*). *Timed ω-regular expressions over an alphabet* Σ (also referred to as *ω-Σ-expressions*) are constructed from (finitary) regular expressions using the following families of rules.

(1) If φ is a Σ-expression, then φ^ω is an ω-Σ-expression.
(2) If φ is a Σ-expression and ψ, ψ_1, ψ_2 are ω-Σ-expressions, then $\varphi \cdot \psi$ and $\psi_1 \vee \psi_2$ are ω-Σ-expressions.
(3) If ψ_1, ψ_2 are ω-Σ-expressions and ψ_0 is an ω-Σ_0 expression for some alphabet Σ_0, and $\theta : \Sigma_0 \to \Sigma$ is a renaming, then $\psi_1 \wedge \psi_2$ and $\theta(\psi_0)$ are ω-Σ-expressions.

Expressions formed using rules (1) and (2) are called timed ω-regular expressions and denoted by $\mathcal{E}_\omega(\Sigma)$. If, in addition, rule (3) is applied we call them generalized timed ω-regular expression and denote them by $\mathcal{GE}_\omega(\Sigma)$.

The semantics of these expressions is defined via the function $[\![]\!]_\omega : \mathcal{GE}_\omega(\Sigma) \to 2^{\mathcal{T}_\omega(\Sigma)}$ as:

$$\begin{aligned}
[\![\varphi^\omega]\!]_\omega &= \mathop{\cdot}_{i=1}^{\infty} [\![\varphi]\!] \\
[\![\varphi \cdot \psi]\!]_\omega &= [\![\varphi]\!] \cdot [\![\psi]\!]_\omega \\
[\![\psi_1 \vee \psi_2]\!]_\omega &= [\![\psi_1]\!]_\omega \cup [\![\psi_2]\!]_\omega \\
[\![\psi_1 \wedge \psi_2]\!]_\omega &= [\![\psi_1]\!]_\omega \cap [\![\psi_2]\!]_\omega \\
[\![\theta(\psi)]\!]_\omega &= \theta([\![\psi]\!]_\omega).
\end{aligned}$$

A timed ω-automaton is a tuple $\mathcal{A} = (Q, C, \Delta, \Sigma, s, F)$ where all the components are as in finitary timed automata. An infinite run of the automaton is an infinite sequence of steps

$$(q_0, \mathbf{v}_0) \xrightarrow{z_1} (q_1, \mathbf{v}_1) \xrightarrow{z_2} \cdots$$

such that the sum of the durations of the steps diverges. The *trace* of a run is the ω-time-event sequence $z_1 \cdot z_2 \cdots$. An accepting run is a run starting from the initial configuration $(s, \mathbf{0})$ and visiting F infinitely many times, that is, $q_i \in F$ for infinitely many discrete steps. The ω-language of a timed automaton, $L_\omega(\mathcal{A})$, consists of all the traces of its accepting runs. Note that due to ε-transitions the trace can be a finite sequence.

7.2. FROM ω-EXPRESSIONS TO ω-AUTOMATA. As in the finitary case, the inductive construction is rather straightforward. As a basis, we take the automaton for any finitary-timed regular expression. From the proof of Theorem 2, we can assume that timed regular languages are accepted by automata without transitions outgoing from accepting states. The automaton for φ^ω is similar to that for φ^+. The accepting state is visited infinitely often in the ω-automaton iff infinitely many finite prefixes of the time-event sequence lead from s to f in the finitary automaton. The concatenation of a language and an ω-language, as well as the union of two ω-languages and the renaming are almost identical to the finitary case. Intersection requires some more details, because, unlike finite words which have to reach accepting states of both automata *simultaneously* at the end of the run, the visits of an ω-time-event sequence in such accepting states need not be synchronized. All the constructions are minor adaptations of their untimed analogues (see Thomas [1990]).

Definition 7.3 (*ω-Automata from Expressions*). Let $\mathcal{A} = (Q, C, \Delta, \Sigma, s, F)$ be the timed automaton accepting the language $[\![\varphi]\!]$, and let $\mathcal{A}_1 = (Q_1, C_1, \Delta_1, \Sigma, s_1, F_1)$ and $\mathcal{A}_2 = (Q_2, C_2, \Delta_2, \Sigma, s_2, F_2)$ be the timed ω-automata accepting the ω-languages $[\![\psi_1]\!]_\omega$ and $[\![\psi_2]\!]_\omega$ respectively.

—The automaton for $[\![\varphi^\omega]\!]_\omega$ is $(Q \cup \{f'\}, C, \Sigma, \Delta', s, \{f'\})$ where Δ' is obtained from Δ by adding for each transition $(q, \phi, \rho, a, f) \in \Delta$ with $f \in F$, a new transition (q, ϕ, C, a, f'). Another transition $(f', (x = 0), \emptyset, \varepsilon, s)$, where x is any clock, is also added (if there is no clock in the automaton we should add one).

—The automaton for $[\![\varphi \cdot \psi_2]\!]$ is $(Q \cup Q_2, C \cup C_2, \Delta', \Sigma, s, F_2)$ where Δ' is $\Delta \cup \Delta_2$ augmented with transitions of the form (q, ϕ, C_2, a, s_2) for every transition (q, ϕ, ρ, a, f) in Δ with $f \in F$.

—The automaton for $[\![\psi_1 \vee \psi_2]\!]_\omega$ is $(Q_1 \cup Q_2 \cup \{s\}, C_1 \cup C_2, \Delta, \Sigma, s, F_1 \cup F_2)$, where Δ is constructed from $\Delta_1 \cup \Delta_2$ by adding two ε-transitions $(s, x = 0, \emptyset, \varepsilon, s_i)$, where x is any clock and $i \in \{1, 2\}$ (if there is no clock in the automata we should add one).

—The automaton for $[\![\varphi_1 \wedge \varphi_2]\!]_\omega$ is $(Q_1 \times Q_2 \times \{1, 2, 3\}, C_1 \cup C_2, \Delta, \Sigma, \langle s_1, s_2, 1\rangle, F)$ where Δ is constructed from Δ_1 and Δ_2 in the following way:

 — for every $(q_1, \phi_1, \rho_1, a, q_1') \in \Delta_1$ and $(q_2, \phi_2, \rho_2, a, q_2') \in \Delta_2$ the relation Δ contains the transitions $(\langle q_1, q_2, i\rangle, \phi_1 \wedge \phi_2, \rho_1 \cup \rho_2, a, \langle q_1', q_2', j\rangle)$ whenever $i = 3$ and $j = 1$, or $i \in \{1, 2\}$ and $j = i$, or $i = 1$, $q_1' \in F_1$ and $j = 2$, or $i = 2$, $q_2' \in F_2$ and $j = 3$.

 — for every $(q_1, \phi_1, \rho_1, \varepsilon, q_1') \in \Delta_1$ the relation Δ contains the transitions $(\langle q_1, q_2, i\rangle, \phi_1, \rho_1, \varepsilon, \langle q_1', q_2, j\rangle)$ whenever $i = 3$ and $j = 1$, or $i \in \{1, 2\}$ and $j = i$, or $i = 1$, $q_1' \in F_1$ and $j = 2$;

 — for every $(q_2, \phi_2, \rho_2, \varepsilon, q_2') \in \Delta_2$ the relation Δ contains the transitions $(\langle q_1, q_2, i\rangle, \phi_2, \rho_2, \varepsilon, \langle q_1, q_2', j\rangle)$ $i = 3$ and $j = 1$, or $i \in \{1, 2\}$ and $j = i$, or $i = 2$, $q_2' \in F_2$ and $j = 3$.

 The accepting set is $F = Q_1 \times Q_2 \times \{3\}$.

—The automaton for $[\![\theta(\psi_1)]\!]_\omega$, where $\theta : \Sigma \to \Sigma'$, is $(Q_1, C_1, \Delta, \Sigma', s_1, F_1)$ with Δ obtained from Δ_1 by replacing every transition of the form (q, ϕ, ρ, a, q') in Δ_1 by $(q, \phi, \rho, \theta(a), q')$.

With this construction, we have the first part of Büchi–McNaughton theorem.

THEOREM 7.4 (ω-EXPRESSIONS $\Rightarrow$ ω-AUTOMATA). *Every (generalized) timed ω-regular language can be accepted by a timed ω-automaton.*

7.3. FROM ω-AUTOMATA TO ω-EXPRESSIONS. This construction is based on Theorem 6.11 and on the proof of the untimed theorem (see Büchi [1960] and McNaughton [1966]). We assume that the automaton has gone through all the transformation described in Section 6.2 and also converted in a state-reset form, as described below.

A one-clock timed automaton is *state-reset* if the transitions entering a given state either all reset the clock, or all do not reset it. In order to make a one-clock automaton state-reset, we split every state not satisfying this property into two copies and redirect the resetting incoming transitions to the first state and nonresetting to the second. This transformation can double the number of states and does not affect the language accepted.

Let $\mathcal{A} = (Q, \{x\}, \Delta, \Sigma, s, F)$ be a one-clock ω-automaton. Clearly,

$$L_\omega(\mathcal{A}) = \bigcup_{f \in F} L_\omega(\mathcal{A}_f),$$

where $\mathcal{A}_f = (Q, \{x\}, \Delta, \Sigma, \{s\}, \{f\})$. Hence, it is sufficient to prove regularity for automata with one accepting state $F = \{f\}$. If f is a resetting state, we can use the same expression as in untimed automata:

$$L_\omega(\mathcal{A}_f) = L_{sf} \cdot (L_{ff})^\omega$$

where L_{sf} is the regular language consisting of all time-event sequences leading from s to f and L_{ff} is the regular language consisting of the time-event sequences inducing a cycle from f to f. However, when f is not resetting, this will not work directly because f can be entered with different clock valuations. The following technical lemma introduces several languages related to one-clock automata and states their regularity.

LEMMA 7.5. *Let $\mathcal{A} = (Q, \{x\}, \Delta, \Sigma, s, \{f\})$ be a one-clock automaton with $m \in \mathbb{N}$ being the largest constant appearing in the guards, and let $p, q \in Q$ be two states. The following timed languages are regular:*

—*The language R°_{pq} consisting of traces of all the runs of $\mathcal{A}$ starting in $(p, 0)$ and terminating by a transition to q and including only non-resetting transitions.*

—*The language $R^{\to m}_{pq}$ consisting of traces of all the runs of $\mathcal{A}$ starting in $(p, 0)$ and terminating by a transition to (q, x) with some $x > m$.*

—*The language $R^{m\to}_{pq}$ consisting of traces of all the runs of $\mathcal{A}$ starting in (p, x), $x > m$, never resetting x and terminating by a transition to q.*

PROOF. The regularity proof for the first two is by a straightforward construction of one-clock sub-automata of $\mathcal{A}$ accepting these languages and by application of Theorem 6.10. For the third, we just erase resetting transitions and substitute $m + 1$ instead of x in all the guards and hence transform each of them into either

true or **false**. Note that the expression obtained for this language contains no timing restrictions. □

Suppose now that f is nonresetting. All the accepting runs split into two categories: those with finitely many resets (whose traces form the language L_{fin}) and those with infinitely many resets (language L_∞). We prove regularity of both these languages.

Finitely Many Resets. Let m denote the maximal constant occurring in the guards of $\mathcal{A}$. Any accepting run ξ with finitely many resets eventually stops resetting the clock and hence the clock value crosses m and remains greater than m ever after. Hence, such a run can be decomposed into a prefix containing all the resets and leading for the first time after that to f with $x > m$, and an infinite suffix making cycles from f to f with x always greater than m. Because timing does not play a role after $x > m$, the languages accepted from (f, x) and from (f, x') for $x, x' > m$ are identical and hence we can write:

$$L_{\mathsf{fin}} = R_{sf}^{\to m} \cdot \left(R_{ff}^{m\to}\right)^\omega,$$

which concludes the proof of regularity of L_{fin}.

Infinitely Many Resets. Since f is not a resetting state, such an infinite run should visit infinitely many times a resetting state q. Moreover, there is always a resetting q such that for infinitely many occurrences, there are no resets between q and the next occurrence of f:

$$\begin{array}{l}(s,0) \to \cdots \to (q,0) \overset{\text{no resets}}{\to \cdots \to} \\ (f, x_1) \to \cdots \to (q,0) \overset{\text{no resets}}{\to \cdots \to} (f, x_2) \to \cdots \to (q,0) \cdots\end{array}$$

Conversely, any run admitting such a decomposition is an accepting run of $\mathcal{A}$.

This immediately gives the following expression for L_∞:

$$L_\infty = \bigcup_{q \text{ resetting}} R_{sq} \cdot (R_{qf}^\circ \circ R_{fq})^\omega$$

which concludes the proof. □

Consequently

CLAIM 7.6. *The ω-language accepted by any one-clock automaton is ω-regular.*

This implies:

THEOREM 7.7 (ω-AUTOMATA $\Rightarrow$ ω-EXPRESSIONS). *Every ω-language accepted by a timed ω-automaton can be represented as*

$$\theta\left(\bigwedge_{i=1}^{k} \psi_i\right),$$

where θ is a renaming and ψ_i are timed ω-regular expressions.

And we can conclude:

THEOREM 7.8 (BÜCHI–MCNAUGHTON THEOREM FOR TIMED AUTOMATA). *Timed ω-automata and generalized timed ω-regular expressions have the same expressive power.*

8. *Discussion*

In this section, we summarize the results of this article and compare our approach to other relevant works. In our view, there are three major contributions in this article:

(1) Clean algebraic definitions of timed behaviors as elements of the monoids of time-event sequences or of signals.
(2) The definition of timed regular expressions as a formalism for specifying timed languages.
(3) The main results and their proof techniques that shed some light on the structure of timed automata and timed languages, in particular the separation of clocks and the elimination of $\circ$ and $^{\circledast}$.

The algebraic definitions, we feel, are simple and intuitive as they treat the succession of events and the accumulation of time-passage in a uniform manner using the same monoid operation. In contrast, timed traces consisting of sequences of time-stamped events do not have this nice monoidal intuition. Compare our concatenation of $r \cdot a$ and $s \cdot b$ into $r \cdot a \cdot s \cdot b$ with the concatenation of the timed traces (a, r) and (b, s) into $(a, r), (b, r + s)$.

Our design choices for the expressions are, perhaps, the closest one can get to the spirit of the untimed theory in the sense that the expressions do not refer to *internal mechanisms* or *hidden variables* of an accepting automaton (states and clocks) but only to *externally observable* properties of the languages. The only (unavoidable) deviation from this spirit is the renaming operator. An alternative formalism that does mention clocks explicitly was proposed in Bouyer and Petit [2002] where the authors define regular expressions over an alphabet consisting of tuples of the form (ϕ, a, ρ) corresponding to the transitions of the timed automaton, where ϕ is a condition on clocks and ρ is a reset. For example, the language defined by our expression

$$((\langle \underline{a} \cdot \underline{b} \rangle_3 \cdot \underline{c}) \wedge (\underline{a} \cdot \langle \underline{b} \cdot \underline{c} \rangle_3)$$

will be written in their syntax as

$$(a, x_2 := 0) \cdot (x_1 = 3, b) \cdot (x_2 = 3, c).$$

The formulation and solution of language equations over this alphabet of transitions is as simple as for untimed automata. A similar idea was phrased in Bouyer and Petit [1999] in terms of expressions constructed using a variety of concatenation operators, each corresponding to a subset of clocks being reset (in the case of one-clock automata this boils down to the $\cdot$ and $\circ$ operations). Using these formalisms, intersection and renaming are avoided at the high price of being very close to the timed automata themselves.

An alternative way to get rid of intersection is to use *many-sorted* parentheses, each corresponding to another clock. For example, the above language could be written as

$$\langle a \cdot \lfloor b \rangle_3 \cdot c \rfloor_3.$$

The drawback of this formalism is that its syntax does not admit a simple inductive definition and, likewise, its semantics cannot be inductively defined. Hence, it can be seen as a syntactic sugar for separation of clocks and intersection.

Our result provides a "Computer Science" version of Kleene Theorem: matching the expressive power of the most commonly-accepted automaton-based formalism for real time by a class of regular expressions. Within the algebraic theory of automata, Kleene Theorem is viewed as a (rare) instance of a coincidence between two different notions, *recognizability* and *rationality*. Recognizability of a subset L of a monoid M can be defined in automaton-free terms. Let $\sim$ be the syntactic right congruence associated with L, namely

$$u \sim v \quad \text{iff} \quad \forall w \in M(u \cdot w \in L \Leftrightarrow v \cdot w \in L).$$

The language L is said to be recognizable if $\sim$ has finitely many congruence classes (and, according to Myhill–Nerode Theorem, this is true if and only if L is accepted by a finite automaton). The class of rational subsets of a monoid M is the rational closure of the finite sets, that is, the smallest class containing finite sets and closed under $\cup$, $\cdot$ and *. Kleene Theorem states that for the free monoid Σ^* recognizability and rationality are equivalent (and this is not true for most other monoids of interest).

This work is concerned with the monoid $\mathcal{T}(\Sigma) = \Sigma^* \boxplus \mathbb{R}_+$, for which, due to the density of $\mathbb{R}_+$, these two notions are not very useful. In $\mathbb{R}_+$ the only recognizable subsets are $\emptyset$, $\{0\}$, $\mathbb{R}_+$ and $(0, \infty)$. A language such as $\langle \underline{a} \rangle_1 \cdot b$ has uncountably many right-congruence classes because $r \not\sim s$ for every $r \neq s \in [0, 1]$. These observation were made already in Rabinovich and Trakhtenbrot [1997] and the conclusion is that only "speed-independent" language, i.e. those invariant under "stretching" are recognizable. Such languages can be written using expressions that do not use $\langle . \rangle$ at all or use it only with intervals $[0, \infty)$ or $(0, \infty)$. Hence, recognizability in this sense is not a useful concept for quantitative time.

Similarly, rationality for $\mathbb{R}_+$ and $\mathcal{T}(\Sigma)$ does not coincide with the expressive needs of timing analysis. On one hand, the class of rational subsets of $\mathbb{R}_+$ contains sets consisting of isolated irrational (and even uncomputable) numbers which cannot be expressed nor accepted by timed automata or any other reasonable device. In addition they may contain arithmetical progressions. On the other hand, a very natural subset of $\mathbb{R}_+$ such as $[0, 1]$ is not rational since it cannot be generated from finite sets by a finite number of applications of the algebraic operations.

These two problems can be resolved by considering the rational closure of Σ and the set of all integer bounded variables. This solution eliminates isolated irrational points and allows to express intervals but the expressive power is still very weak: the set $\langle \underline{a} \rangle_{[1,2]} \cdot \langle \underline{b} \rangle_{[2,4]}$ is in the rational closure but the set denoted by $\langle \underline{a} \cdot \underline{b} \rangle_{[3,6]}$ is not. Such sets correspond to one-clock timed automata that reset the clock after each transition (see Dima [2001]). An interesting option for overcoming this limitation is to introduce a new *shuffle* operator, but this is beyond the scope of this article. We may conclude that a Kleene theorem (in the strict algebraic sense) for timed monoids is impossible.

REFERENCES

ALUR, R., AND DILL, D. L. 1994. A Theory of timed automata. *Theor. Comput. Sci. 126*, 183–235.

ARDEN, D. 1960. Delayed-logic and finite-state machines. In *Theory of Computing Machine Design*. Univ. of Michigan Press, pp. 1–35.

ASARIN, E. 1998. Equations on timed languages. In *Hybrid Systems: Computation and Control*, T. A. Hezinger and S. Sastry, Eds. Lecture Notes in Computer Science, vol. 1386. Springer-Verlag, New York, pp. 1–12,

ASARIN, E., CASPI, P., AND MALER, O. 1997. A Kleene theorem for timed automata, In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science* (*LICS'97*) (Warsaw, Poland, June). IEEE Computer Society Press, Los Alamitos, Calif., pp. 160–171.

BOUYER, P., AND PETIT, A. 1999. Decomposition and composition of timed automata. In *Proceedings of the 26th International Colloquium Automata, Languages, and Programming* (*ICALP'99*), J. Wiedermann, P. van Emde Boas, and M. Nielsen, Eds. Lecture Notes in Computer Science, vol. 1644. Springer-Verlag, New York, pp. 210–219.

BOUYER, P., AND PETIT, A. 1999. A Kleene/Büchi-like theorem for clock languages. *J. Autom. Lang. Combin.*, to appear.

BÜCHI, J. 1960. A decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, E. Nagel, Ed. Stanford University Press, Stanford, Calif.

CONWAY, J. H. 1971. *Regular Algebra and Finite Machines*, Chapman & Hall, London, England.

DIMA, C. 2001. Real-time automata. *J. Autom. Lang. Combin. 6*, 1, pp. 3–24.

HERRMANN, P. 1999. Renaming is necessary in timed regular expressions. In *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science* (*FSTTCS '1999*) (Chennai, India, Dec. 13–15). Lecture Notes in Computer Science, vol. 1738. Springer-Verlag, New York, pp. 47–59.

HOWIE, J. M. 1995. *Fundamentals of Semigroup Theory*. Clarendon Press, Oxford, England.

LARSEN, K. G., PETTERSSON, P., AND YI, W. 1997. UPPAAL in a nutshell. *Int. J. Softw. Tools Tech. Trans. 1*, 1–2 (Oct.), 134–152.

LI, X., TAO, Z., JIANMIN, H., JIANHUA, Z., AND GUOLIANG, Z. 1998. Hybrid regular expressions. In *Hybrid Systems: Computation and Control*, T. A. Hezinger and S. Sastry, Eds. Lecture Notes in Computer Science, vol. 1386. Springer-Verlag, New York, pp. 384–399.

MALER, O., MANNA, Z., AND PNUELI, A. 1992. From timed to hybrid systems. In *Real Time: Theory in Practice*, J. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, Eds. Lecture Notes in Computer Science, vol. 600. Springer-Verlag, New York, pp. 447–484.

MCNAUGHTON, R. 1966. Testing and generating infinite sequences by a finite automaton. *Inf. Contr. 9*, 521–530.

MCNAUGHTON, R., AND YAMADA, H. 1960. Regular expressions and state graphs for automata. *IRE Trans. Elect. Comput. EC-9*, 39–47.

RABINOVICH, A., AND TRAKHTENBROT, B. 1997. From finite automata toward hybrid systems (extended abstract). In *Fundamentals of Computation Theory, 11th International Symposium, FCT '97*, B. S. Chlebus and L. Czaja, Eds. Lecture Notes in Computer Science, vol. 1279. Springer-Verlag, New York, pp. 411–422.

THOMAS, W. 1990. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, J. V. Leeuwen, Ed. Vol. B. Elsevier, Amsterdam, 133–191.

TRAKHTENBROT, B. 1995. Origins and metamorphoses of the trinity: Logics, nets, automata. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science* (*LICS'95*). IEEE Computer Society Press, Los Alamitos, Calif., pp. 506–507.

YOVINE, S. 1997. Kronos: A verification tool for real-time systems. *Int. J. Softw. Tools Tech. Trans. 1*, 1–2 (Oct.), 123–133.

RECEIVED JUNE 2001; REVISED DECEMBER 2001; ACCEPTED DECEMBER 2001