Temporal Logics, Automata, and Classical Theories for Defining Real-Time Languages*

T.A. Henzinger[1], J.-F. Raskin[1,2], and P.-Y. Schobbens[3]

[1]Electrical Engineering and Computer Sciences,
University of California, Berkeley, USA

[2]Computer Science Department,
Université Libre de Bruxelles, Belgium

[3]Computer Science Institute,
Université de Namur, Belgium

November 10, 1999

Abstract

A specification formalism for reactive systems defines a class of ω-languages. We call a specification formalism *fully decidable* if it is constructively closed under boolean operations and has a decidable satisfiability (nonemptiness) problem. There are two important, robust classes of ω-languages that are definable by fully decidable formalisms. The ω-REGULAR LANGUAGES are definable by finite automata, or equivalently, by the Sequential Calculus. The COUNTER-FREE ω-REGULAR LANGUAGES are definable by temporal logic, or equivalently, by the first-order fragment of the Sequential Calculus. The gap between both classes can be closed by finite counting (using automata connectives), or equivalently, by projection (existential second-order quantification over letters).

A specification formalism for real-time systems defines a class of *timed* ω-languages, whose letters have real-numbered time stamps. Two popular ways of specifying timing constraints rely on the use of clocks, and on the use of time bounds for temporal operators. However, temporal logics with clocks or time bounds have undecidable satisfiability problems, and finite automata with clocks (so-called *timed automata*) are not closed under complement. Therefore, two fully decidable restrictions of these formalisms have been proposed. In the first case, clocks are restricted to *event clocks*, which measure distances to immediately preceding or succeeding events only. In the second case, time bounds are restricted to *nonsingular intervals*, which cannot specify the exact punctuality of events. We show that the resulting classes of timed ω-languages are robust, and we explain their relationship.

First, we show that temporal logic with event clocks defines the same class of timed ω-languages as temporal logic with nonsingular time bounds, and we identify a first-order monadic theory that also defines this class. Second, we show that if the ability of finite counting is added to these formalisms, we obtain the class of timed ω-languages that are definable by finite automata with event clocks, or equivalently, by a restricted second-order extension of the monadic theory. Third, we show that if projection is added, we obtain the class of timed ω-languages that are definable by timed automata, or equivalently, by the full second-order extension of the monadic theory. These results identify three robust classes of timed ω-languages, of which the third, while popular, is not definable by a fully decidable formalism. By contrast, the first two classes are definable by fully decidable formalisms from temporal logic, from automata theory, and from monadic logic. Since the gap between these two classes can be

*NSF CAREER award CCR-9501708,DARPA (NASA) grant NAG2-1214, DARPA (Wright-Patterson AFB),grant F33615-C-98-3614, ARO MURI grant DAAH-04-96-1-0341, by the Belgian National Fund for Scientific Research (FNRS), by the European Commission under WGs Aspire and Fireworks, and by Belgacom.

closed by finite counting, we dub them the *timed ω-regular languages* and the *timed counter-free ω-regular languages*, respectively.

Contents

1 Introduction

A run of a reactive system produces an infinite sequence of events. A property of a reactive system, then, is an ω-language containing the infinite event sequences that satisfy the property. There is a very pleasant expressive equivalence between modal logics, classical logics, and finite automata for defining ω-languages [Büc62, Kam68, GPSS80, Wol82]. Let LTL stand for the propositional linear temporal logic with next and until operators, and let $\mathsf{Q\text{-}TL}$ and $\mathsf{E\text{-}TL}$ stand for the extensions of LTL with propositional quantifiers and grammar (or automata) connectives, respectively. Let ML_1 and ML_2 stand for the first-order and second-order monadic theories of the natural numbers with successor and comparison (also called $\mathsf{S1S}$ or the Sequential Calculus). Let BA stand for Büchi automata. Then we obtain the following two levels of expressiveness:

	Languages	Temporal logics	Monadic theories	Finite automata
1	*counter-free ω-regular*	LTL	ML_1	
2	*ω-regular*	$\mathsf{Q\text{-}TL} = \mathsf{E\text{-}TL}$	ML_2	BA

For example, the LTL formula $\Box(p \to \Diamond q)$, which specifies that every p event is followed by a q event, is equivalent to the ML_1 formula $(\forall i)(p(i) \to (\exists j \geq i)q(j))$ and to a Büchi automaton with two states. The difference between the first and second levels of expressiveness is the ability of automata to count. A counting requirement, for example, may assert that all even events are p events, which can be specified by the $\mathsf{Q\text{-}TL}$ formula $(\exists q)(q \wedge \Box(q \leftrightarrow \bigcirc\neg q) \wedge \Box(q \to p))$.

We say that a formalism is *positively decidable* if it is constructively closed under positive boolean operations, and satisfiability (emptiness) is decidable. A formalism is *fully decidable* if it is positively decidable and also constructively closed under negation (complement). All of the formalisms in the above table are fully decidable. The temporal logics and Büchi automata are less succinct formalisms than the monadic theories, because only the former satisfiability problems are elementarily decidable.

A run of a real-time system produces an infinite sequence of time-stamped events. A property of a real-time system, then, is a set of infinite time-stamped event sequences. We call such sets *timed ω-languages.* If all time stamps are natural numbers, then there is again a very pleasant expressive equivalence between modal logics, classical logics, and finite automata [AH93]. Specifically, there are two natural ways of extending temporal logics with timing constraints. The Metric Temporal Logic $\mathsf{MetricTL}$ (also called MTL [AH93]) adds time bounds to temporal operators; for example, the $\mathsf{MetricTL}$ formula $\Box(p \to \Diamond_{=5}\, q)$ specifies that every p event is followed by a q event such that the difference between the two time stamps is exactly 5. The Clock Temporal Logic $\mathsf{ClockTL}$ (also called TPTL [AH94]) adds clock variables to LTL; for example, the time-bounded response requirement from above can be specified by the $\mathsf{ClockTL}$ formula $\Box(p \to (x := 0)\Diamond(q \wedge x = 5))$, where x is a variable representing a clock that is started by the quantifier $(x := 0)$. Interestingly, over natural-numbered time, both ways of expressing timing constraints are equally expressive. Moreover, by adding the ability to count, we obtain again a canonical second level of expressiveness. Let $\mathsf{TimeFunctionMLR}$ stand for the monadic theory of the natural numbers extended with a unary function symbol that maps event numbers to time stamps, and let TA (Timed Automata) be finite automata with clock variables. In the following table, the formalisms are annotated with the superscript $\mathbb{N}$ to emphasize the fact that all time stamps are natural numbers:

	Languages	Temporal logics	Monadic theories	Finite automata
1	*$\mathbb{N}$-timed counter-free ω-regular*	$\mathsf{MetricTL}^{\mathbb{N}} = \mathsf{ClockTL}^{\mathbb{N}}$	$\mathsf{TimeFunctionMLR}_1^{\mathbb{N}}$	
2	*$\mathbb{N}$-timed ω-regular*	$\mathsf{Q\text{-}MetricTL}^{\mathbb{N}} = \mathsf{Q\text{-}ClockTL}^{\mathbb{N}} = \mathsf{E\text{-}MetricTL}^{\mathbb{N}} = \mathsf{E\text{-}ClockTL}^{\mathbb{N}}$	$\mathsf{TimeFunctionMLR}_2^{\mathbb{N}}$	$\mathsf{TA}^{\mathbb{N}}$

Once again, all these formalisms are fully decidable, and the temporal logics and finite automata with timing constraints are elementarily decidable.

If time stamps are real instead of natural numbers, then the situation seems much less satisfactory. Several positively and fully decidable formalisms have been proposed, but no expressive equivalence results were known for fully decidable formalisms [AH92]. The previously known results are listed in the following table, where the omission of superscripts indicates that time stamps are real numbers:

Temporal logics	Monadic theories	Finite automata
Fully decidable		
MetricIntervalTL [AFH96] EventClockTL [RS97]		
		REventClockTA [AFH94]
Positively decidable		
$\mathsf{LTL}^+ + \mathsf{TA}$ [Wil94]	$\mathcal{L}d^{\leftrightarrow}$ [Wil94]	TA [AD94]
Fully undecidable		
MetricTL [AH93]		
ClockTL [AH94]	$\mathsf{TimeFunctionMLR}_1$ [AH93]	
	$\mathsf{TimeFunctionMLR}_2$	

On one hand, the class of Timed Automata is unsatisfactory, because over real-numbered time it is only positively decidable: $\mathbb{R}$-timed automata are not closed under complement, and the corresponding temporal and monadic logics (and regular expressions [?]) have no negation operator. On the other hand, the classes of Metric and Clock Temporal Logics (as well as monadic logic with a time function), which include negation, are unsatisfactory, because over real-numbered time their satisfiability problems are undecidable. Hence several restrictions of these classes have been studied.

1. The first restriction concerns the style of specifying timing constraints using time-bounded temporal operators. The Metric-Interval Logic MetricIntervalTL (also called MITL [AFH96]) is obtained from MetricTL by restricting the time bounds on temporal operators to nonsingular intervals. For example, the MetricIntervalTL formula $\Box(p \rightarrow \Diamond_{[4,6]}\, q)$ specifies that every p event is followed by a q event such that the difference between the two time stamps is at least 4 and at most 6. The restriction to nonsingularity prevents the specification of the exact real-numbered time difference 5 between events.

2. The second restriction concerns the style of specifying timing constraints using clock variables. The Event-Clock Logic EventClockTL (also called SCL [RS97]) and Event-Clock Automata REventClockTA are obtained from ClockTL and TA, respectively, by restricting the use of clocks to refer to the times of previous and next occurrences of events only. For example, if y_q is a clock that always refers to the time difference between now and the next q event, then the EventClockTL formula $\Box(p \rightarrow y_q = 5)$ specifies that every p event is followed by a q event such that the difference between time stamps of the p event and the first subsequent q event is exactly 5. A clock such as y_q, which is permanently linked to the next q event, does not need to be started explicitly, and is called an *event clock*. The restriction to event clocks prevents the specification of time differences between a p event and any subsequent (rather than the first subsequent) q event.

Both restrictions lead to pleasing formalisms that are fully (elementarily) decidable and have been shown sufficient in practical applications. However, nothing was known about the relative expressive powers of these two independent approaches, and so the question which sets of timed ω-languages deserve the labels "$\mathbb{R}$-timed counter-free ω-regular" and "$\mathbb{R}$-timed ω-regular" remained open.

In this paper, we show that MetricIntervalTL and EventClockTL are equally expressive, and by adding the ability to count, as expressive as REventClockTA. This result is quite surprising, because (1) over real-numbered time, unrestricted MetricTL is known to be strictly less expressive than unrestricted ClockTL [AH93], and (2) the nonsingularity restriction (which prohibits exact time differences but allows the comparison of unrelated events) is very different in flavor from the event-clock restriction (which allows exact time differences but prohibits the comparison of unrelated events). Moreover, the expressive equivalence of Metric-Interval and Event-Clock logics reveals a robust picture of canonical specification formalisms for real-numbered time that parallels the untimed case and the case of natural-numbered time. We complete this picture by characterizing both the counter-free and the counting levels of expressiveness also by fully decidable monadic theories, called $\mathsf{MinMaxML}_1$ and $\mathsf{MinMaxML}_2$. These are first-order and second-order monadic theories of the real numbers with integer addition, comparison, and (besides universal and existential quantification) two first-order quantifiers that determine the first time and the last time at which a formula is true. Our results, which are summarized in the following table, suggest that we have identified two classes of ω-languages with real-numbered time stamps that may justly be called "$\mathbb{R}$-timed counter-free ω-regular" and "$\mathbb{R}$-timed ω-regular":

	Languages	Temporal logics	Monadic theories	Finite automata
Fully decidable				
1	$\mathbb{R}$-*timed counter-free* ω-*regular*	MetricIntervalTL = EventClockTL	$\mathsf{MinMaxML}_1$	
2	$\mathbb{R}$-*timed* ω-*regular*	Q-MetricIntervalTL = Q-EventClockTL = E-MetricIntervalTL = E-EventClockTL	$\mathsf{MinMaxML}_2$	REventClockTA

Finally, we explain the gap between the $\mathbb{R}$-timed ω-regular languages and the languages definable by positively decidable formalisms such as timed automata. We show that the richer class of languages is obtained by closing the $\mathbb{R}$-timed ω-regular languages under projection. (It is unfortunate, but well-known [AFH94] that we cannot nontrivially have both full decidability and closure under projection in the case of real-numbered time.) The complete picture, then, results from adding the following line to the previous table (projection, or outermost existential quantification, is indicated by P-):

Positively decidable				
3	*projection-closed* $\mathbb{R}$-*timed* ω-*regular*	P-EventClockTL	$\mathsf{P\text{-}MinMaxML}_2 = \mathcal{L}d^{\leftrightarrow}$	P-REventClockTA = TA

The rest of this paper is organized as follows. The real-time models that we are considering in this papers are presented in section 2. Two real-time logics and a classical theories are introduced in section 3. Their relative expressive power is studied in details: those logics are shown to be expressively equivalent and they identify the "counter-free regular realt-ime languages". Section 4 contains the definition and a study of the properties of the recursive event clock automata. It is shown that the class of languages recognized by recursive event-clock automata strictly subsumes the class of "counter-free regular realt-ime languages" and we call this class the "(full) regular real-time languages". Section 5 studies the relation that exists between the logical and automata theoretic formalisms. Furthermore, we show how to bridge the gap that exists between "counter-free regular realt-ime languages" and "(full) regular real-time languages". Finally some conclusions are drawn in a last section.

2 Real-Time Models

In this paper, we consider real-time behaviors that are modeled by a function κ that assign to each point of the real line a state description. Thus the function κ at each $t \in \mathbb{R}^+$ indicates the state $\kappa(t)$ in which the system is at that time t. We make two assumptions about the function κ:

Finite Variability (also called Non Zenoness) The function κ has the finite variability property: during each finite interval of time I, the value of κ only changes a finite number of time. This assumption avoids the so-called zeno paradox: the system does an infinite number of actions into a finite amount of time.

Finite State Systems The number of different discrete states, i.e. the size of the set of possible states that the system can reach is finite.

The *finite state* assumption allows us to use a finite set of propositions to describe those states. The codomain of the function κ is then the powerset of $\mathcal{P}$, noted $2^{\mathcal{P}}$. The *finite variability* assumption allows us to represent the function κ using two infinite sequences: one infinite sequence of subsets of $\mathcal{P}$ to represent the discrete part of the behavior of the system, and an infinite sequence of intervals of time indicating for each state when the system was in that state. We call those pairs of sequences, timed state sequences and define them formally in the sequel. Later, we use also the notion of *finite variable formula*, it simply means that the truth value of the formula change only a finite number of times in every bounded interval of time.

Definition 2.1 (Intervals of Time) An *interval* (of time) $I \subseteq \mathbb{R}^+$ is a convex nonempty subset of the nonnegative reals. And interval I is bounded (above) by $b \in \mathbb{R}^+$ if for all $t \in I$, $t \leq b$. Due to our definition, every interval is bounded below by 0. By completeness of the real numbers, every bounded interval has a *least upper bound*, that we call its *right bound*. If the interval is unbounded, we conventionally define its least upper bound as ∞. Symmetrically, each interval has a *greatest lower bound*, that we also call its left bound. In each case, the bound can be either included in I, this is noted by a square bracket, or excluded from I, this is noted by a round parenthesis. We have thus the six following possibilities:

1. closed finite: $[l, r]$ with $l, r \in \mathbb{R}^+$ and $l \leq r$. Specially, when $l = r$, the interval is called *singular*;
2. left open, right closed: $(l, r]$ with $l, r \in \mathbb{R}^+$ and $l < r$;
3. left closed, right open: $[l, r)$ with $l, r \in \mathbb{R}^+$ and $l < r$;
4. open: (l, r) with $l, r \in \mathbb{R}^+$ and $l < r$;
5. left closed, infinite: $[l, \infty)$ with $l \in \mathbb{R}^+$;
6. left open, infinite: (l, ∞) with $l \in \mathbb{R}^+$.

Two intervals I and J are *adjacent* if the right bound of I is equal to the left bound of J, and either I is right-open and J is left-closed or I is right-closed and J is left-open. Thus two adjacent intervals are disjoint. □

Notation 2.2 (Intervals) The left bound of interval I is noted $l(I)$, the right end bound of interval I is noted $r(I)$. Given $t \in \mathbb{R}^+$, we freely use notation such as $t + I$ for the interval $\{t' \mid \text{exists } t'' \in I \text{ with } t' = t + t''\}$, and $t > I$ for the constraint "$t > t'$ for all $t' \in I$." □

Definition 2.3 (Interval Sequence) An *interval sequence* $\overline{I} = I_0, I_1, \ldots$ is a finite or infinite sequence of bounded intervals so that for all $i \geq 0$, the intervals I_i and I_{i+1} are adjacent. We say that the interval sequence $\overline{I}$ *covers* the interval $\bigcup_{i \geq 0} I_i$. If $\overline{I}$ covers $[0, \infty)$, then $\overline{I}$ partitions the nonnegative real line so that every bounded subset of $\mathbb{R}^+$ is contained within a finite union of elements from the partition. □

We are now in position to define our notion of continuous models called *timed state sequence* and noted TSS.

Definition 2.4 (Timed State Sequence) The set of states is called Σ. A *timed state sequence* $\kappa = (\overline{\sigma}, \overline{I})$ over Σ is a pair that consists of an trace $\overline{\sigma} = \sigma_0 \sigma_1 \ldots \sigma_n \ldots$ over Σ and an infinite interval sequence $\overline{I} = I_0 I_1 \ldots I_n \ldots$ that covers $[0, \infty)$. □

Equivalently, the timed state sequence κ can be viewed as a function from $\mathbb{R}^+$ to Σ, indicating for each time $t \in \mathbb{R}^+$ a state $\kappa(t)$.

We now introduce two different type of real-time languages: the anchored and floating real-time languages. The notion of anchored languages is the classical one, the notion floating languages is not classical and is needed for technical reasons in the sequel of this paper.

Definition 2.5 (Pointwise Real-Time ω-Languages) A *pointwise anchored real-time ω-language* is a set of timed traces. A *pointwise floating real-time ω-language* is a set of pairs (θ, i) where θ is a timed trace and $i \geq 0$ is a position. □

In the sequel we consider that $\Sigma = 2^{\mathcal{P}}$ and we need notion related to the addition and suppression of propositions in the set on which a timed state sequences is defined. It is why we introduce the notion the notion of $\mathcal{P}'$-extension and $\mathcal{P}'$-projection of a TSS.

Definition 2.6 ($\mathcal{P}'$-Extension of a TSS) Given a TSS $\kappa = (\overline{\sigma}, \overline{I})$ defined on the set of propositions $\mathcal{P}$, a set of propositions $\mathcal{P}'$, such that $\mathcal{P} \cap \mathcal{P}' = \emptyset$, $\kappa' = (\overline{\sigma}', \overline{I}')$ is a *$\mathcal{P}'$-extension* of κ if κ' is defined on the set of propositions $\mathcal{P} \cup \mathcal{P}'$ and for all position $i \geq 0$: (i) $\sigma'_i \cap \mathcal{P} = \sigma_i$, that is, state description σ'_i and σ_i agree on the set of propositions $\mathcal{P}$, and (ii) $I'_i = I_i$, the real-time information attached to the state descriptions is similar in the two TSS. We note $\kappa \uparrow \mathcal{P}'$ the set of $\mathcal{P}'$-extension of κ.

Definition 2.7 ($\mathcal{P}'$-Projection of a TSS) Given a TSS $\kappa = (\overline{\sigma}, \overline{I})$ defined on the set of propositions $\mathcal{P}$, a set of propositions $\mathcal{P}' \subseteq \mathcal{P}$, κ' is the *$\mathcal{P}'$-projection* of κ, if κ' is defined on the set of propositions $\mathcal{P}'$ and for every positions $i \geq 0$: (i) $\sigma'_i = \sigma_i \cap \mathcal{P}'$ that is σ'_i and σ_i agree on the value of propositions in $\mathcal{P}'$, and (ii) $I'_i = I_i$, the real-time information attached to the state descriptions is similar in the two TSS. In the sequel, we note $\kappa \downarrow \mathcal{P}'$ the $\mathcal{P}'$-projection of κ.

As we consider continuous models, it will turn out, in section 4.4.2, that the notion of limit closure is useful:

Definition 2.8 (Limit Closure - Literal) Given a set of propositions $\mathcal{P}$, we define its *limit closure*, noted $\mathsf{Limit}(\mathcal{P})$, as the following set $\{p, \overrightarrow{p}, \overleftarrow{p} \mid p \in \mathcal{P} \cup \{\top\}\}$, $\overrightarrow{p}$ is called the *future limit of* p and $\overleftarrow{p}$ is called the *past limit of* p. In what follows, we call the elements of $\mathsf{Limit}(\mathcal{P})$ literals. In what follows, we use $\mathcal{L}$, $\mathcal{L}_1$, $\mathcal{L}_2$, ..., to denote limit closure sets. □

Later, we will generalize the use of limit. We will apply the limit not only to propositions but also to atomic clock constraints.

Definition 2.9 (Satisfaction Relation) We write $(\kappa, t) \models \phi$, where ϕ is a proposition, an literal, an atomic clock constraint or more generally a formula, read "ϕ is satisfied at time t of the TSS κ". We define the semantics for propositions $p \in \mathcal{P}$ and for the special symbol $\top$ (true):

- $(\kappa, t) \models p$ iff $p \in \kappa(t)$;
- $(\kappa, t) \models \top$ for all time $t \in \mathbb{R}^+$.

□

The rules for more general formulas will be given later, we now give the semantics for the limit literals:

Definition 2.10 (Future and Past Limits Semantics) The truth value of the future limit of $p \in \mathcal{P} \cup \{\top\}$ along a TSS κ is defined by the following clause:

$$(\kappa, t) \models \overrightarrow{p} \text{ iff for all time } t_1 > t \text{ there exists a time } t_2 \text{, such that } t < t_2 < t_1 \text{ and } (\kappa, t_2) \models p;$$

The truth value of the past limit of $p \in \mathcal{P} \cup \{\top\}$ along a TSS κ is defined by the following clause:

$$(\kappa, t) \models \overleftarrow{p} \text{ iff for all time } t_1 < t \text{ there exists a time } t_2 \geq 0 \text{, such that } t_1 < t_2 < t \text{ and } (\kappa, t_2) \models p.$$

Note that $\overrightarrow{\top}$ is always equivalent to $\top$. In time 0, $\overleftarrow{\top}$ is equivalent to $\bot$ and equivalent to $\top$ elsewhere. □

Intuitively, the future (resp. past) limit of p at time t allows us to access the truth value of p just after (resp. before) time t.

We now define a serie a useful properties of TSS:

Definition 2.11 ($\Psi - \mathsf{Fine}$ TSS) Given a set of finite variable formulas Ψ, we say that a TSS $\kappa = (\sigma_0, I_0)(\sigma_1, I_1)\ldots$ is $\Psi - \mathsf{Fine}$ iff for all positions $i \geq 0$, for all formula $\psi \in \Psi$, for any time $t_1, t_2 \in I_i$, we have that $(\kappa, t_1) \models \psi$ iff $(\kappa, t_2) \models \psi$, that is, the truth value of the formula ψ does not change inside the intervals of κ. □

Definition 2.12 (Alternating-TSS) We say that a TSS $\kappa = (s_0, I_0)(s_1, I_1)\ldots$ is *alternating* iff

1. I_0 is the singular interval $[0, 0]$;
2. for all even positions i, I_i is a singular interval, and
3. for all odd positions i, I_i is a open interval.

□

Definition 2.13 (Hintikka Property) Given a set of formulas Ψ, a timed state sequence κ has the *Hintikka property for* Ψ, iff

1. κ is defined on a set of propositions that contains the set $\mathcal{P}$ of propositions appearing in the formulas of Ψ and the following set of *hintikka propositions* $\mathcal{P}^{\Psi} = \{p_\psi \mid \psi \in \Psi\}$, that is, a hintikka proposition for each formula of the set Ψ,
2. for every time $t \in \mathbb{R}^+$, $(\kappa, t) \models p_\psi$ iff $(\kappa, t) \models \psi$, that is, a hintikka proposition is true along a Hintikka sequence at time t if and only if its associated formula is true at time t.

□

When manipulating a Hintikka TSS $\kappa = (\overline{\sigma}, \overline{I})$, we sometimes write $\phi \in \sigma_i$ instead of $p_\phi \in \sigma_i$ in order to simplify the notations.

Definition 2.14 (Equivalent TSS) Two TSS κ^1, κ^2 are equivalent iff $\kappa^1(t) = \kappa^2(t)$ for all time $t \in \mathbb{R}^+$, that is, if the two TSS define the same function from the positive real numbers to state descriptions. □

So two TSS are equivalent if they only differ by the way they split the real line is into intervals.

Definition 2.15 (Refinement of TSS) A TSS $\kappa^1 = (\overline{\sigma}^1, \overline{I}^1)$ is a refinement of a TSS $\kappa^2 = (\overline{\sigma}^2, \overline{I}^2)$, noted $\kappa^1 \leq \kappa^2$ iff there exists a surjective function $f : \mathbb{N} \to \mathbb{N}$ such that:

- for all positions $j \geq 0$, $\sigma^2_j = \sigma^1_{f(j)}$;
- for all positions $i \geq 0$, $I^1_i = \bigcup\{I^2_j \mid f(j) = i\}$

In what follows, we also say that κ^2 is coarser than κ^1. □

Note that TSS κ^1 is a refinement of the TSS κ^2 then κ^1 and κ^2 are equivalent.

Lemma 2.16 (Refinability of TSS) *For every* TSS κ *and every set of formula* Ψ *with the finite variability property, there exists a* TSS κ' *such that (i)* $\kappa' \leq \kappa$*, that is,* κ' *is a refinement of* κ *and (ii)* κ' *is* $\Psi - \mathsf{Fine}$.

Note also that:

Lemma 2.17 (Refinement and Fine-TSS) *For every set of formulas* Ψ*, every refinement* κ' *of a* $\Psi - \mathsf{Fine}$ TSS κ *is* $\Psi - \mathsf{Fine}$.

And thus this refinement can be alternating:

Lemma 2.18 *For every* TSS κ*, there exists a refinement* κ' *of* κ*, i.e.* $\kappa' \leq \kappa$ *that is alternating.*

In the sequel we use sets of literals to label locations of automata. We will need the notion of singular and open set of literals. Intuitively, a singular literal describes an instantaneous, unstable situation and thus cannot hold during an open interval of time. Here are their definitions:

Definition 2.19 (Singular-Open Set of Literals) A set of literals $\Xi \subseteq \mathcal{L}$ is said *singular* iff one of the two following properties of Ξ is verified

- there exist literals $a, \overrightarrow{a} \in \mathcal{L}$ such that $a \in \Xi$ and $\overrightarrow{a} \notin \Xi$, or, $a \notin \Xi$ and $\overrightarrow{a} \in \Xi$;
- there exist literals $a, \overleftarrow{a} \in \mathcal{L}$ such that $a \in \Xi$ iff $\overleftarrow{a} \notin \Xi$, or, $a \notin \Xi$ iff $\overleftarrow{a} \in \Xi$. An set of literals $\Xi \subseteq \mathcal{L}$ is said *open* iff it is not singular.

□

Lemma 2.20 *Let* I *be a non singular interval. If* Ξ *is singular, then for all* κ*, there exists* $t \in I$ *such that* $(\kappa, t) \not\models \Xi$. □

3 The Counter-Free Regular Real-Time ω-Languages

3.1 Introduction

In this section, we introduce two real-time logics and a classical theory for defining real-time properties. We study their expressive power in details and show that they all identify the same class of real-time languages that we call the *counter-free regular real-time languages*. Before, we recall the definition of two qualitative time formalisms and review a theorem about their relative expressive power introduced by Kamp.

3.2 Qualitative Formalisms

3.2.1 The Temporal Logic of the Reals: LTR

We review in this section a temporal logic that is evaluated over continuous models. That temporal logic is called the *temporal logic of the reals*, noted LTR, and has been proposed by Pnueli et al in [BKP86]. We recall its syntax and semantics.

Definition 3.1 (LTR-Syntax) The formulas of LTR are built from propositional symbols, boolean connectives, and the temporal "until" and "since" operators:

$$\phi \ ::= \ p \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{S}_I \phi_2$$

where p is a proposition, ϕ, ϕ_1 and ϕ_2 are well-formed LTR formulas. □

Definition 3.2 (LTR-Semantics) The LTR formula ϕ *holds* at time $t \in \mathbb{R}^+$ of the timed state sequence κ, denoted $(\kappa, t) \models \phi$, according to the following definition:

- $(\kappa, t) \models p$ iff $p \in \kappa(t)$;
- $(\kappa, t) \models \phi_1 \wedge \phi_2$ iff $(\kappa, t) \models \phi_1$ and $(\kappa, t) \models \phi_2$;
- $(\kappa, t) \models \neg\phi$ iff not $(\kappa, t) \models \phi$;
- $(\kappa, t) \models \phi_1 \mathcal{U} \phi_2$ iff exists a real $t' > t$ with $(\kappa, t') \models \phi_2$, and for all reals $t'' \in (t, t')$, we have $(\kappa, t'') \models \phi_1 \vee \phi_2$;
- $(\kappa, t) \models \phi_1 \mathcal{S} \phi_2$ iff exists a real $t' \in [0, t)$ with $(\kappa, t') \models \phi_2$, and for all reals $t'' \in (t', t)$, we have $(\kappa, t'') \models \phi_1 \vee \phi_2$.

□

Definition 3.3 (LTR-Languages) The *anchored language* defined by an LTR formula ϕ is the set of TSS $\kappa \in \mathsf{TSS}(2^{\mathcal{P}_\phi})$, such that $(\kappa, 0) \models \phi$, this set is noted $\mathsf{AncLang}(\phi)$. The *floating language* defined by an LTR formula ϕ is the set of pairs (κ, t) with $\kappa \in \mathsf{TSS}(2^{\mathcal{P}_\phi})$ and $t \in \mathbb{R}^+$ such that $(\kappa, t) \models \phi$, this set is noted $\mathsf{FloatLang}(\phi)$. □

3.2.2 The First-Order Monadic Logic over the Reals: MLR_1

We now review the definition of the first-order monadic logic of the reals. We recall its syntax and semantics.

Definition 3.4 (MLR_1-Syntax) The formulas of the *first- order monadic logic over the reals* MLR_1 are generated by the following grammar:

$$\Phi ::= p(x) \mid x_1 = x_2 \mid x_1 < x_2 \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid \exists x \cdot \Phi$$

where $x, x_1, x_2 \in X$ are position variables (first-order variable), $p \in \mathcal{P}$ is an unary predicate and Φ, Φ_1, Φ_2 are well-formed MLR_1 formulas. We say that a formula Φ of MLR_1 is closed if it does not contain any free position variable. □

Definition 3.5 (Valuation) A valuation for the set of first-order variables X is a mapping $\alpha : X \to \mathbb{R}^+$ assigning a nonnegative real number value to each variable $x \in X$. We note $\alpha[y \mapsto t]$ the mapping that extend the mapping α for the variable y and maps y on the value $t \in \mathbb{R}^+$. □

Definition 3.6 (MLR_1**-Semantics**) The semantics of an MLR_1 formula Φ is evaluated in pair (κ, α) where κ is a TSS and α is a valuation for the free variables appearing in Φ according to the following rules:

$(\kappa, \alpha) \models q(x)$ iff $q \in \kappa(\alpha(x))$;
$(\kappa, \alpha) \models x_1 = x_2$ iff $\alpha(x_1) = \alpha(x_2)$;
$(\kappa, \alpha) \models x_1 < x_2$ iff $\alpha(x_1) < \alpha(x_2)$;
$(\kappa, \alpha) \models \neg\Phi$ iff $(\kappa, \alpha) \not\models \Phi$;
$(\kappa, \alpha) \models \Phi_1 \vee \Phi_2$ iff $(\kappa, \alpha) \models \Phi_1$ or $(\kappa, \alpha) \models \Phi_2$;
$(\kappa, \alpha) \models \exists x \cdot \Phi$ iff there exists a value $t \in \mathbb{R}^+$ such that $(\kappa, \alpha[x \mapsto t]) \models \Phi$.

□

Definition 3.7 (MLR_1**-language**) The *anchored language* defined by a closed MLR_1 formula Φ is the set of TSS $\kappa \in \mathsf{TSS}(2^{\mathcal{P}_\Phi})$, such that $\kappa \models \Phi$, this set is noted $\mathsf{AncLang}(\Phi)$. The *floating language* defined by an MLR_1 formula $\Phi(x)$, with one free variable x is the set of pairs (κ, t) with $\kappa \in \mathsf{TSS}(2^{\mathcal{P}_\Phi})$ and $t \in \mathbb{R}^+$ such that $(\kappa, [x \mapsto t]) \models \Phi$, this set is noted $\mathsf{FloatLang}(\Phi)$. □

3.2.3 Expressive Equivalence: $\mathsf{LTR} = \mathsf{MLR}_1$

Kamp has proved, see [Kam68], that the expressiveness equivalence result between temporal logic and the first-order monadic logic is also valid in the case of continuous interpretations:

Theorem 3.8 ($\mathsf{LTR} = \mathsf{MLR}_1$) *The logics* LTR *and* MLR_1 *are equally expressive: given an* LTR *formula* ϕ*, there always exists a closed formula* Φ *of* MLR_1 *such that* $\mathsf{AncLang}(\phi) = \mathsf{AncLang}(\Phi)$*, and conversely: given an* MLR_1 *formula* Φ*, there always exists a formula* ϕ *of* LTR *such that* $\mathsf{AncLang}(\Phi) = \mathsf{AncLang}(\phi)$*. Furthermore, given an* LTR *formula* ϕ*, there always exists a formula* $\Phi(x)$ *with one free variable* x *of* MLR_1 *such that* $\mathsf{FloatLang}(\phi) = \mathsf{FloatLang}(\Phi(x))$*, and conversely: given an* MLR_1 *formula* $\Phi(x)$ *with one free variable* x*, there always exists a formula* ϕ *of* LTR *such that* $\mathsf{FloatLang}(\Phi) = \mathsf{FloatLang}(\phi)$. □

3.3 Two Real-Time Temporal Logics

3.3.1 The Metric Interval Temporal Logic: $\mathsf{MetricIntervalTL}$

Here, we recall the definition of the logic $\mathsf{MetricIntervalTL}$ [AFH91, AFH96]. This logic is a syntactical restriction of the undecidable real-time logic $\mathsf{MetricTL}$ [AH90]. The logic $\mathsf{MetricIntervalTL}$ prohibits the specification of punctuality constraints by allowing only subscripts in real-time operators that are nonsingular intervals. This restriction makes the formalism decidable.

Definition 3.9 ($\mathsf{MetricIntervalTL}$**-Syntax**) The formulas of $\mathsf{MetricIntervalTL}$ [AFH96] are built from propositional symbols, boolean connectives, and time-bounded "until" and "since" operators:

$$\phi ::= p \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \phi_1 \widehat{\mathcal{U}}_I\, \phi_2 \mid \phi_1 \widehat{\mathcal{S}}_I\, \phi_2$$

where p is a proposition and I is a *nonsingular* interval whose finite endpoints are nonnegative integers. □

Note that we use hats in the syntax of the temporal operators above in order to deferentiate them from the operator of LTR that have a slightly different semantics in the qualitative case. We also define an interesting subset of $\mathsf{MetricIntervalTL}$, called $\mathsf{MetricIntervalTL}_{0,\infty}$:

Definition 3.10 ($\mathsf{MetricIntervalTL}_{0,\infty}$**-Syntax**) The formulas of the fragment $\mathsf{MetricIntervalTL}_{0,\infty}$ are defined as for $\mathsf{MetricIntervalTL}$, except that the interval I must either have the left endpoint 0, or be unbounded; in these cases I can be replaced by an expression of the form $\sim c$, for a nonnegative integer constant c and $\sim \in \{<, \leq, \geq, >\}$. □

Definition 3.11 ($\mathsf{MetricIntervalTL}$ **Continuous Semantics**) The $\mathsf{MetricIntervalTL}$ formula ϕ *holds* at time $t \in \mathbb{R}^+$ of the timed state sequence κ, denoted $(\kappa, t) \models \phi$, according to the following definition

$(\kappa, t) \models p$ iff $p \in \kappa(t)$
$(\kappa, t) \models \phi_1 \wedge \phi_2$ iff $(\kappa, t) \models \phi_1$ and $(\kappa, t) \models \phi_2$
$(\kappa, t) \models \neg\phi$ iff not $(\kappa, t) \models \phi$
$(\kappa, t) \models \phi_1 \widehat{\mathcal{U}}_I\, \phi_2$ iff exists a real $t' \in (t + I)$ with $(\kappa, t') \models \phi_2$, and for all reals $t'' \in (t, t')$, we have $(\kappa, t'') \models \phi_1$
$(\kappa, t) \models \phi_1 \widehat{\mathcal{S}}_I\, \phi_2$ iff exists a real $t' \in (t - I)$ with $(\kappa, t') \models \phi_2$, and for all reals $t'' \in (t', t)$, we have $(\kappa, t'') \models \phi_1$

□

We now introduce some useful abbreviations:

Definition 3.12 (MetricIntervalTL-Abbreviations) For the future:

- $\widehat{\Diamond}_I \phi = \top \widehat{\mathcal{U}}_I \phi$, "eventually in the future within interval I";
- $\widehat{\Box}_I = \neg\widehat{\Diamond}_I \neg\phi$, "always in the future within interval I".

Symetrically, for the past:

- $\widehat{\diamondminus}_I \phi = \top \widehat{\mathcal{S}}_I \phi$, "eventually in the past within interval I";
- $\widehat{\boxminus}_I = \neg\widehat{\diamondminus}_I \neg\phi$, "always in the past within interval I".

Definition 3.13 (MetricIntervalTLContinuous Languages) The MetricIntervalTL formula ϕ *defines* the *anchored language* that contains all timed state sequences κ with $(\kappa, 0) \models \phi$. As usual, we note this language AncLang(ϕ). The MetricIntervalTL formula ϕ *defines* the *floating language* that contains all pairs (κ, t) with $(\kappa, t) \models \phi$. As usual, we note this language FloatLang(ϕ). □

Example 3.14 The MetricIntervalTL formula $\widehat{\Box}_{(0,1)}(p \rightarrow \widehat{\Diamond}_{[1,2]}\, q)$ asserts when evaluated in time t, that every p-state, in the interval $t + (0, 1)$, is followed by a q-state at a time difference of at least 1 and at most 2 time units. □

The complexity of the satisfiability and validity problems for MetricIntervalTL and its fragments $\mathsf{MetricIntervalTL}_{0,\infty}$ are given in the next theorem.

Theorem 3.15 [AFH96] *The satisfiability and validity problems for* MetricIntervalTL *are* ExpSpace-Complete. *The satisfiability and validity problems for* $\mathsf{MetricIntervalTL}_{0,\infty}$ *are* PSpace-Complete. □

Interestingly, the complexity of the satisfiability and validity problems for $\mathsf{MetricIntervalTL}_{0,\infty}$ are easier that for the full logic.

3.3.2 The Logic of Event Clocks: EventClockTL

The formulas of EventClockTL are built from propositional symbols, boolean connectives, the temporal "until" and "since" operators, and two real-time operators: at any time t, the *history operator* $\triangleleft_I \phi$ asserts that ϕ was true last time in the interval $t - I$, and the *prophecy operator* $\triangleright_I \phi$ asserts that ϕ will be true next time in the interval $t + I$.

Definition 3.16 (Continuous-EventClockTL-Syntax) The formulas of (continuous) EventClockTL for timed state sequences are generated by the following grammar:

$$\phi \ ::= \ p \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{S} \phi_2 \mid \triangleleft_I \phi \mid \triangleright_I \phi$$

where p is a proposition and I is an interval whose finite endpoints are nonnegative integers. □

We can now define how to evaluate the truth value of an EventClockTL formula along timed state sequences.

Definition 3.17 (Continuous-$\mathsf{EventClockTL}$-Semantics) Let ϕ be an (continuous) $\mathsf{EventClockTL}$ formula and let τ be a timed state sequence whose propositional symbols contain all propositions that occur in ϕ. The formula ϕ *holds* at time $t \in \mathbb{R}^+$ of τ, denoted $(\tau, t) \models \phi$, according to the following definition:

$(\tau, t) \models p$ iff $p \in \tau(t)$
$(\tau, t) \models \phi_1 \vee \phi_2$ iff $(\tau, t) \models \phi_1$ or $(\tau, t) \models \phi_2$
$(\tau, t) \models \neg\phi$ iff not $(\tau, t) \models \phi$
$(\tau, t) \models \phi_1 \mathcal{U} \phi_2$ iff exists a real $t' > t$ with $(\tau, t') \models \phi_2$, and for all reals $t'' \in (t, t')$, we have $(\tau, t'') \models \phi_1 \vee \phi_2$
$(\tau, t) \models \phi_1 \mathcal{S} \phi_2$ iff exists a real $t' < t$ with $(\tau, t') \models \phi_2$, and for all reals $t'' \in (t', t)$, we have $(\tau, t'') \models \phi_1 \vee \phi_2$
$(\tau, t) \models \lhd_I \phi$ iff exists a real $t' < t$ with $t' \in (t - I)$ and $(\tau, t') \models \phi$, and for all reals $t'' < t$ with $t'' > (t - I)$, not $(\tau, t'') \models \phi$
$(\tau, t) \models \rhd_I \phi$ iff exists a real $t' > t$ with $t' \in (t + I)$ and $(\tau, t') \models \phi$, and for all reals $t'' > t$ with $t'' < (t + I)$, not $(\tau, t'') \models \phi$

□

Note that the temporal and real-time operators are defined in a strict manner; that is, they do not constrain the current state. Non strict operators are easily defined from their strict counterparts.

Example 3.18 $\Box(p \rightarrow \rhd_{\leq 5} q)$: a p position is always followed by a q position within 5 time units. Such a formula specifies a maximal distance between a request p and its response q. Such a property is called a *bounded time response*. Here, it assumes that only one request can be outstanding. $p \wedge \Box(p \rightarrow \rhd_{=1} p)$: this formula asserts that p is true every integer time unit. Such a formula allows the specifier to define *periodicity of events*. Here p can model the tick of an ideal clock, that ticks every time unit. $\Box((\lhd_{=3} q) \rightarrow p)$. This formula asserts that if the last q position is exactly distant of 3 time units then p must be true now. It is a typical *time-out requirement*.
□

We now give a definition of the real-time languages that a $\mathsf{EventClockTL}$ formula is defining.

Definition 3.19 (Continuous-$\mathsf{EventClockTL}$-Languages) The (continuous) $\mathsf{EventClockTL}$ formula ϕ defines the anchored language $\mathsf{AncLang}(\phi) = \{\kappa \mid (\kappa, 0) \models \phi\}$, that is the set of timed state sequences that satisfy ϕ at their initial position. The (continuous) $\mathsf{EventClockTL}$ formula ϕ defines the floating language $\mathsf{FloatLang}(\phi) = \{(\kappa, t) \mid (\kappa, t) \models \phi\}$, that is the set of pairs (timed state sequence, time) where ϕ is verified.
□

3.4 A First-Order Classical Theory: $\mathsf{MinMaxML}_1$

In the sequel, we use p, q, and r for (finite variable) monadic predicates over the nonnegative reals, and t, t_1, and t_2 for first-order variables over $\mathbb{R}^+$.

Definition 3.20 ($\mathsf{MinMaxML}_1$-Syntax) The formulas of the *First-Order Real-Time Sequential Calculus*, noted $\mathsf{MinMaxML}_1$, are generated by the following grammar:

$$\begin{array}{lll}\Phi & ::= & p(t) \mid t_1 \sim t_2 \mid \\ & & (\mathsf{Min}\ t_1)(t_1 > t_2 \wedge \Psi(t_1)) \sim (t_2 + c) \mid \\ & & (\mathsf{Max}\ t_1)(t_1 < t_2 \wedge \Psi(t_1)) \sim (t_2 - c) \mid \\ & & \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid (\exists t)\Phi\end{array}$$

where $\Psi(t_1)$ is a $\mathsf{MinMaxML}_1$ formula that contains no free occurrences of first-order variables other than t_1, where c is a nonnegative integer constant, and $\sim \in \{<, \leq, =, \geq, >\}$. □

The truth value of a $\mathsf{MinMaxML}_1$ formula Φ is evaluated over a pair (κ, α) that consists of a timed state sequence κ whose propositional symbols contain all monadic predicates of Φ, and a valuation α that maps each free first-order variable of Φ to a nonnegative real. By $\alpha_{[t\mapsto v]}$ we denote the valuation that agrees with α on all variables except t, which is mapped to the value v. We first define for each $\mathsf{MinMaxML}_1$ term ς a value $\mathsf{Val}_{\kappa,\alpha}(\varsigma)$, which is either a nonstandard real or undefined. Intuitively, the term $(\mathsf{Min}\ t_1)(t_1 > t_2 \wedge \Psi(t_1))$ denotes the smallest value greater than t_2 that satisfies the formula Ψ. If there is no value greater than t_2 that satisfies Ψ, then the term $(\mathsf{Min}\ t_1)(t_1 > t_2 \wedge \Psi(t_1))$ denotes the undefined value $\perp$. If Ψ is satisfied throughout a left-open interval with left endpoint $v > t_2$, then the term $(\mathsf{Min}\ t_1)(t_1 > t_2 \wedge \Psi(t_1))$ denotes the nonstandard real number v^+. Similarly, the term $(\mathsf{Max}\ t_1)(t_1 < t_2 \wedge \Psi(t_1))$ denotes the greatest value smaller than t_2 that satisfies Ψ. [1] Formally:

Definition 3.21 ($\mathsf{MinMaxML}_1$-Term Values) The value of a term ς in the TSS κ and valuation α, denoted $\mathsf{Val}_{\kappa,\alpha}(\varsigma)$, is defined by the following rules:

$$\mathsf{Val}_{\kappa,\alpha}(t) = \alpha(t)$$
$$\mathsf{Val}_{\kappa,\alpha}(t+c) = \alpha(t) + c$$
$$\mathsf{Val}_{\kappa,\alpha}(t-c) = \begin{cases} \alpha(t) - c & \text{if } \alpha(t) \geq c \\ \perp & \text{otherwise} \end{cases}$$
$$\mathsf{Val}_{\kappa,\alpha}((\mathsf{Min}\ t_1)(t_1 > t_2 \wedge \Psi(t_1)) =$$
$$\begin{cases} v & \text{if } (\kappa, \alpha_{[t_1\mapsto v]}) \models (t_1 > t_2 \wedge \Psi(t_1)), \\ & \quad \text{and for all } v' < v, \text{ not } (\kappa, \alpha_{[t_1\mapsto v']}) \models (t_1 > t_2 \wedge \Psi(t_1)) \\ v^+ & \text{if for all } v' > v, \text{ exists } v'' < v' \text{ with } (\kappa, \alpha_{[t_1\mapsto v'']}) \models (t_1 > t_2 \wedge \Psi(t_1)), \\ & \quad \text{and for all } v' \leq v, \text{ not } (\kappa, \alpha_{[t_1\mapsto v']}) \models (t_1 > t_2 \wedge \Psi(t_1)) \\ \perp & \text{if for all } v \geq 0, \text{ not } (\kappa, \alpha_{[t_1\mapsto v]}) \models (t_1 > t_2 \wedge \Psi(t_1)) \end{cases}$$
$$\mathsf{Val}_{\kappa,\alpha}((\mathsf{Max}\ t_1)(t_1 < t_2 \wedge \Psi(t_1)) =$$
$$\begin{cases} v & \text{if } (\kappa, \alpha_{[t_1\mapsto v]}) \models (t_1 < t_2 \wedge \Psi(t_1)), \\ & \quad \text{and for all } v' > v, \text{ not } (\kappa, \alpha_{[t_1\mapsto v']}) \models (t_1 < t_2 \wedge \Psi(t_1)) \\ v^- & \text{if for all } v' < v, \text{ exists } v'' > v' \text{ with } (\kappa, \alpha_{[t_1\mapsto v'']}) \models (t_1 < t_2 \wedge \Psi(t_1)), \\ & \quad \text{and for all } v' \geq v, \text{ not } (\kappa, \alpha_{[t_1\mapsto v']}) \models (t_1 < t_2 \wedge \Psi(t_1)) \\ \perp & \text{if for all } v \geq 0, \text{ not } (\kappa, \alpha_{[t_1\mapsto v]}) \models (t_1 < t_2 \wedge \Psi(t_1)) \end{cases}$$

□

Now we can define the satisfaction relation for $\mathsf{MinMaxML}_1$ formulas:

Definition 3.22 ($\mathsf{MinMaxML}_1$-semantics) The following rules define when a formula is satisfied by a TSS κ and a valuation α:

$$(\kappa, \alpha) \models p(t) \text{ iff } p \in \kappa(\alpha(t))$$
$$(\kappa, \alpha) \models t_1 \sim t_2 \text{ iff } \mathsf{Val}_{\kappa,\alpha}(t_1) \sim \mathsf{Val}_{\kappa,\alpha}(t_2), \text{ with } \sim \in \{<, \leq, =, \geq, >\}$$
$$(\kappa, \alpha) \models \Phi_1 \vee \Phi_2 \text{ iff } (\kappa, \alpha) \models \Phi_1 \text{ or } (\kappa, \alpha) \models \Phi_2$$
$$(\kappa, \alpha) \models \neg\Phi \text{ iff not } (\kappa, \alpha) \models \Phi$$
$$(\kappa, \alpha) \models (\exists t)\Phi \text{ iff exists } v \geq 0 \text{ with } (\kappa, \alpha_{[t\mapsto v]})\Phi$$

□

A $\mathsf{MinMaxML}_1$ formula is *closed* iff it contains no free occurrences of first-order variables. Every closed $\mathsf{MinMaxML}_1$ formula defines an anchored real-time language:

Definition 3.23 ($\mathsf{MinMaxML}_1$-Anchored Language) Every closed $\mathsf{MinMaxML}_1$ formula Φ *defines* an anchored real-time ω-language, namely, the set of real-time state sequences κ such that $(\kappa, \emptyset) \models \Phi$.

And every $\mathsf{MinMaxML}_1$ formula with one free variable defines a floating real-time language:

[1] Note that although the terms take their value in non standard real numbers plus undefined, quantifiers only range over the real numbers.

Definition 3.24 ($\mathsf{MinMaxML}_1$-Floating Language) Every $\mathsf{MinMaxML}_1$ formula Φ with one free first-order variable t_1 *defines a floating real-time ω-language*, namely, the set of pairs (κ, t) such that $(\kappa, [t_1 \mapsto t]) \models \Phi$.

Example 3.25 ($\mathsf{MinMaxML}_1$ formula)

$$(\forall t_1)(p(t_1) \to (\exists t_2)(t_2 > t_1 \wedge q(t_2) \wedge (\mathsf{Min}\ t_3)(t_3 > t_2 \wedge r(t_3)) = t_2 + 5))$$

asserts that every p-state is followed by a q-state that is followed by an r-state after, but no sooner than, 5 time units.

We will show in the next section that the formalism that we have defined in this section is decidable.

3.5 Expressiveness Results

Remember that in section 3.2.3, we have recalled a result proved by Kamp that states the expressive equivalence between the temporal logic of the reals, LTR, and the first-order monadic logic over the reals, MLR_1, see theorem 3.8. We will use this result in the sequel to establish the same theorem about the relative expressive power of $\mathsf{MinMaxML}_1$ and $\mathsf{EventClockTL}$.

3.5.1 $\mathsf{EventClockTL} = \mathsf{MinMaxML}_1$

We first prove that $\mathsf{EventClockTL}$ is at least as expressive as $\mathsf{MinMaxML}_1$. To prove that result, we use theorem 3.8 and reason on the level of $\mathsf{MinMaxML}_1$ formulas. The level of a $\mathsf{MinMaxML}_1$ formula is defined as follows:

Definition 3.26 (level of $\mathsf{MinMaxML}_1$ Formulas) The *level* of a $\mathsf{MinMaxML}_1$ formula ϕ, noted $\mathsf{level}(\phi)$, is defined as follows:

- $\mathsf{level}(q(t)) = 0$, where q is a monadic predicate;
- $\mathsf{level}(t_1 \sim t_2) = 0$, where t_1, t_2 are first order variables;
- $\mathsf{level}(\Phi_1 \vee \Phi 2) = \mathsf{Maximum}(\mathsf{level}(\Phi_1), \mathsf{level}(\Phi_2))$;
- $\mathsf{level}(\neg\Phi) = \mathsf{level}(\Phi)$;
- $\mathsf{level}(\exists t \cdot \Phi(t)) = \mathsf{level}(\Phi(t))$;
- $\mathsf{level}(\mathsf{Max}_{t_2} \cdot t_2 < t_1 \wedge \Phi(t_2) \sim t_1 - c) = 1 + \mathsf{level}(\Phi(t_2))$;
- $\mathsf{level}(\mathsf{Min}_{t_2} \cdot t_2 > t_1 \wedge \Phi(t_2) \sim t_1 + c) = 1 + \mathsf{level}(\Phi(t_2))$;

So the level of a $\mathsf{MinMaxML}_1$ formula is the number of imbrications of $\mathsf{Min} - \mathsf{Max}$ quantifiers in the formula. □

We now prove the following lemma:

Lemma 3.27 ($\mathsf{MinMaxML}_1 \subseteq \mathsf{EventClockTL}$) *For every formula $\Phi(t_1)$ of $\mathsf{MinMaxML}_1$ with one free variable t_1, there exists a congruent formula Φ^T of $\mathsf{EventClockTL}$, that is for every TSS κ and every time $t \in \mathbb{R}^+$: $(\kappa, [t_1 \mapsto t]) \models \Psi(t_1)$ iff $(\kappa, t) \models \Psi^T$.*

Proof. We reason by induction on the level of formula.

- *Base case.* Let $\Phi(t_1)$ be such that $\mathsf{level}(\Phi(t_1)) = 0$. In that case, the formula $\Phi(t_1)$ does not contains any $\mathsf{Min} - \mathsf{Max}$ quantifier and thus $\Phi(t_1)$ is a ML_1 formula. By theorem 3.8, there exists an congruent LTR formula Φ^T. As LTR is a subset of $\mathsf{EventClockTL}$, Φ^T is an $\mathsf{EventClockTL}$ formula.

- *Induction case.* Let $\Phi(t_1)$ be such that $\mathsf{level}(\Phi(t_1)) = i$. By induction hypothesis, we are able to construct for every level_j, with $j < i$, formula Ψ of $\mathsf{MinMaxML}_1$, a congruent $\mathsf{EventClockTL}$ formula Ψ^T. We now show that we can also do it for level_i formulas. By definition of the level of a $\mathsf{MinMaxML}_1$ formula, we know that for every subformula of the form:

  - $\mathsf{Max}_{t_2} \cdot [t_2 < t_1 \wedge \Psi(t_2)] \sim t_1 - c$
  - $\mathsf{Min}_{t_2} \cdot [t_2 > t_1 \wedge \Psi(t_2)] \sim t_1 + c$

  $\Psi(t_2)$ is at most of level_{i-1} and by induction hypothesis, can be expressed in $\mathsf{EventClockTL}$ by a congruent formula Ψ^T. Also, by definition of the semantics of $\mathsf{Min} - \mathsf{Max}$ and $\rhd_{\sim c}, \lhd_{\sim c}$, we have the following:

  C.1 $(\kappa, t) \models \rhd_{\sim c} \Psi^T$ iff $(\kappa, [t_1 \mapsto t]) \models \mathsf{Min}_{t_2} \cdot t_2 > t_1 \wedge \Psi(t_2) \sim t_1 + c$

  C.2 $(\kappa, t) \models \lhd_{\sim c} \Psi^T$ iff $(\kappa, [t_1 \mapsto t]) \models \mathsf{Max}_{t_2} \cdot t_2 < t_1 \wedge \Psi(t_2) \sim t_1 - c$

  It remains us to show that the entire formula $\mathsf{MinMaxML}_1$ formula $\Phi(t)$ can be expressed in $\mathsf{EventClockTL}$. We do this by first transforming $\Phi(t)$ as follows: every formula of the form $\mathsf{Min}_{t_2} \cdot t_2 > t_1 \wedge \Psi(t_2) \sim t_1 + c$, $\mathsf{Max}_{t_2} \cdot t_2 < t_1 \wedge \Psi(t_2) \sim t_1 - c$ is replaced by a fresh monadic predicate p^Ψ, we note this formula $\widetilde{\Phi(t)}$ and $\mathcal{P}^\Psi$ the set of fresh monadic predicates that we have used to obtain $\widetilde{\Phi(t)}$. We know that $\widetilde{\Phi(t)}$ is a ML_1 formula over the monadic predicates of $\mathcal{P} \cup \mathcal{P}^\Psi$. By theorem 3.8, we can compute a congruent formula $\widetilde{\Phi}^T$ of LTR. To obtain the desired $\mathsf{EventClockTL}$ formula, it remains us to replace every fresh propositions of p^Ψ in $\widetilde{\Phi}^T$ by Ψ^T (as given by the clauses C1 and C2 above) to obtain the desired formula Φ^T.

□

We now show that the reverse also holds.

Lemma 3.28 ($\mathsf{EventClockTL} \subseteq \mathsf{MinMaxML}_1$) *For every formula ϕ of $\mathsf{EventClockTL}$, there exists a congruent formula ϕ^T with one free variable t_1 of $\mathsf{MinMaxML}_1$, that is for every TSS κ and every time $t \in \mathbb{R}^+$: $(\kappa, [t_1 \mapsto t]) \models \phi^T(t_1)$ iff $(\kappa, t) \models \phi$.*

Proof. We do a classical reasoning on the structure of formulas.

- *Base case.* ϕ is the proposition p. Then ϕ^T is simply $p(t_1)$.
- *Induction case.* By induction hypothesis, we can construct for each subformula ϕ_1, ϕ_2 of $\mathsf{EventClockTL}$, the congruent formulas ϕ_1^T and ϕ_2^T of $\mathsf{MinMaxML}_1$. We show that for each construct of $\mathsf{EventClockTL}$ that are applied to ϕ_1 and ϕ_2, we are able to construct the desired formula of $\mathsf{MinMaxML}_1$:

  - for $\phi = \neg\phi_1$, we take $\phi^T = \neg\phi_1^T(t_1)$;
  - for $\phi = \phi_1 \vee \phi_2$, we take $\phi^T = \phi_1^T(t_1) \vee \phi_2^T(t_1)$;
  - for $\phi = \phi_1 \mathcal{U} \phi_2$, we take $\exists t_2 > t_1 \cdot (\phi_2^T(t_2) \wedge \forall t_3 \cdot t_1 < t_3 < t_2 \cdot \phi_1^T(t_3) \vee \phi_2^T(t_3))$
  - for $\phi = \phi_1 \mathcal{S} \phi_2$, we take $\exists t_2 \cdot 0 \leq t_2 < t_1 \cdot (\phi_2^T(t_2) \wedge \forall t_3 \cdot t_2 < t_3 < t_1 \cdot \phi_1^T(t_3) \vee \phi_2^T(t_3))$
  - for $\phi = \rhd_{\sim c} \phi_1$, we take $\mathsf{Min}_{t_2}[t_1 < t_2 \wedge \phi_1^T(t_2)] \sim t_1 + c$;
  - for $\phi = \lhd_{\sim c} \phi_1$, we take $\mathsf{Max}_{t_2}[t_2 < t_1 \wedge \phi_1^T(t_2)] \sim t_1 - c$.

□

The two previous lemma allow us to derive the following theorem that states the equivalent expressive power of the logics $\mathsf{EventClockTL}$ and $\mathsf{MinMaxML}_1$:

Theorem 3.29 *The floating and anchored real-time ω-regular languages definable by the logic* $\mathsf{EventClockTL}$ *and* $\mathsf{MinMaxML}_1$ *are identical.*

The lemma 3.27 allows us to derive the following decidability results for $\mathsf{MinMaxML}_1$:

Theorem 3.30 ($\mathsf{MinMaxML}_1$-Decidability) *The satisfiability and validity problems of the logic* $\mathsf{MinMaxML}_1$ *are decidable and in* NonElem.

3.5.2 $\mathsf{EventClockTL} = \mathsf{MetricIntervalTL}$

We now turn to the relation that exists between the logic $\mathsf{EventClockTL}$ and the logic $\mathsf{MetricIntervalTL}$.

We first define the fragment $\mathsf{EventClockTL}_{0,\infty}$ of $\mathsf{EventClockTL}$. We will use it in the following proofs.

Definition 3.31 ($\mathsf{EventClockTL}_{0,\infty}$) The formulas of the fragment $\mathsf{EventClockTL}_{0,\infty}$ of $\mathsf{EventClockTL}$ are the formulas that only use real-time operators $\rhd_I$, $\lhd_I$ where: either $l(I)=0$ or $r(I)=\infty$. □

The semantics of $\mathsf{EventClockTL}_{0,\infty}$ formulas is as for full $\mathsf{EventClockTL}$. The following lemma expresses that $\mathsf{EventClockTL}_{0,\infty}$ is expressively complete:

Lemma 3.32 ($\mathsf{EventClockTL} = \mathsf{EventClockTL}_{0,\infty}$) *For every formula of $\phi \in \mathsf{EventClockTL}$ we can construct a congruent formula ϕ^T of the fragment $\mathsf{EventClockTL}_{0,\infty}$, that is for every* TSS *κ, for every time $t \in \mathbb{R}^+$, $(\kappa,t) \models \phi$ iff $(\kappa,t) \models \phi^T$.*

Proof. We reason by induction on the structure of formulas:

- *Base case.* Let $\phi = p$. Then $\phi \in \mathsf{EventClockTL}_{0,\infty}$.
- *Induction case.* The boolean cases and temporal cases are trivial. Let us consider the formula $\rhd_I\psi$, with $l(I) \neq 0$ and $r(I) \neq \infty$. By induction hypothesis, we have $\psi^T \in \mathsf{EventClockTL}_{0,\infty}$. We note I_1 the interval $\{t > 0 \mid \exists t' \in I \text{ and } t \leq t'\}$, and I_2 the interval $\{t > 0 \mid \forall t' \in I,\, t < t'\}$. By definition of ϕ, we know that those two intervals are non-empty, as $l(I) > 0$ and $r(I) < \infty$ and their bounds are integer numbers, and further that $l(I_1) = l(I_2) = 0$. It is easy to see that the formula $\rhd_{I_1}\psi^T \wedge \rhd_{I_2}\psi^T$ is congruent to ϕ and in $\mathsf{EventClockTL}_{0,\infty}$. The case for the operator $\lhd$ is similar and left to the reader.

□

We now prove that the fragment $\mathsf{MetricIntervalTL}_{0,\infty}$ is at least as expressive as the logic $\mathsf{EventClockTL}$.

Lemma 3.33 ($\mathsf{EventClockTL} \subseteq \mathsf{MetricIntervalTL}_{0,\infty}$) *For every formula ϕ of* $\mathsf{EventClockTL}$*, there exists a congruent formula ϕ^T of* $\mathsf{MetricIntervalTL}_{0,\infty}$*, that is for every* TSS *κ and every time $t \in \mathbb{R}^+$: $(\kappa,t) \models \phi^T$ iff $(\kappa,t) \models \phi$.*

Proof. By lemma 3.32, we know that $\mathsf{EventClockTL}_{0,\infty}$ is equally expressive to $\mathsf{EventClockTL}$. Thus it is sufficient to show that $\mathsf{EventClockTL}_{0,\infty} \subseteq \mathsf{MetricIntervalTL}_{0,\infty}$. We reason by induction on the structure of formulas. In the sequel, ϕ belongs to $\mathsf{EventClockTL}_{0,\infty}$ and ϕ^T denotes the congruent $\mathsf{MetricIntervalTL}_{0,\infty}$ formula.

- *Base case.* The formula ϕ is the proposition p then $\phi^T = p$;
- *Induction cases*: by induction hypothesis, ϕ_1 and ϕ_2 are translated by ϕ_1^T and ϕ_2^T. Here are the different cases:

- for $\phi = \phi_1 \vee \phi_2$, we take $\phi^T = \phi_1^T \vee \phi_2^T$;
- for $\phi = \neg\phi_1$, we take $\phi^T = \neg\phi_1^T$;
- for $\phi = \phi_1 \mathcal{U} \phi_2$, we take $\phi^T = (\phi_1^T \vee \phi_2^T)\widehat{\mathcal{U}}_{(0,\infty)}\phi_2^T$;
- for $\phi = \phi_1 \mathcal{S} \phi_2$, we take $\phi^T = (\phi_1^T \vee \phi_2^T)\widehat{\mathcal{S}}_{(0,\infty)}\phi_2^T$;
- $\phi = \rhd_J \phi_1$ with $l(J) = 0$. Note that the operator $\rhd$ is irreflexive so we can make the hypothesis that $0 \notin J$. We distinguish the case where the first ϕ_1-interval in the future is left closed from the case where it is left open. The two situations can be distinguished by the following $\mathsf{MetricIntervalTL}_{0,\infty}$ formula: $\neg\phi_1^T \widehat{\mathcal{U}}_{(0,\infty)}\phi_1^T$.
  * In the case that the former formula is verified then the following ϕ_1-interval is left closed and we can check that $\rhd_J\phi_1$ is verified by checking the following $\mathsf{MetricIntervalTL}_{0,\infty}$ formula: $\neg\phi_1^T \widehat{\mathcal{U}}_J \phi_1^T$.
  * In the second case, the first ϕ_1-interval is left open and then we check that $\bigcirc\phi_1^T \vee \neg\phi_1^T \widehat{\mathcal{U}}_{(0,l(J))}\bigcirc\phi_1^T$ where $\bigcirc\phi_1^T$ denotes $\bot\widehat{\mathcal{U}}_{(0,\infty)}\phi_1^T$ and means that ϕ_1^T is true just after the present time. Let us note that $l(J)$ is excluded as we check the event $\bigcirc\phi_1^T$ and not the event ϕ_1^T.

  This gives the following translation rule:

$$\begin{aligned} \rhd_J\phi_1 = &\wedge \top\widehat{\mathcal{U}}_{(0,\infty)}\phi_1^T \\ &\wedge \neg\phi_1^T\widehat{\mathcal{U}}_{(0,\infty)}\phi_1^T \rightarrow \neg\phi_1^T\widehat{\mathcal{U}}_J\phi_1^T \\ &\wedge \neg(\neg\phi_1^T\widehat{\mathcal{U}}_{(0,\infty)}\phi_1^T) \rightarrow \bigcirc\phi_1^T \vee \neg\phi_1^T\widehat{\mathcal{U}}_{(0,r(J))}\bigcirc\phi_1^T \end{aligned}$$

- $\phi = \rhd_J \phi_1$ with $l(J) \neq 0$. And thus $r(J) = \infty$ as $\phi \in \mathsf{EventClockTL}_{0,\infty}$. Here also, we distinguish the case where the first ϕ_1-interval in the future is left closed from the case where it is left open. We obtain the following translation rule:

$$\begin{aligned} \rhd_J\phi_1 = &\wedge \top\widehat{\mathcal{U}}_{(0,\infty)}\phi_1^T \\ &\wedge \neg\phi_1^T\widehat{\mathcal{U}}_{(0,\infty)}\phi_1^T \rightarrow \neg\phi_1^T\widehat{\mathcal{U}}_J\phi_1^T \\ &\wedge \neg(\neg\phi_1^T\widehat{\mathcal{U}}_{(0,\infty)}\phi_1^T) \rightarrow \neg\phi_1^T\widehat{\mathcal{U}}_{[l(J),\infty)}(\neg\phi_1^T \wedge\bigcirc\phi_1^T) \end{aligned}$$

- $\phi = \lhd_J \phi_1$ with $l(j) = 0$. By a similar reasoning we obtain:

$$\begin{aligned} \lhd_J\phi_1 = &\wedge \top\widehat{\mathcal{S}}_{(0,\infty)}\phi_1^T \\ &\wedge \neg\phi_1^T\widehat{\mathcal{S}}_{(0,\infty)}\phi_1^T \rightarrow \neg\phi_1^T\widehat{\mathcal{S}}_J\phi_1^T \\ &\wedge \neg(\neg\phi_1^T\widehat{\mathcal{S}}_{(0,\infty)}\phi_1^T) \rightarrow \ominus\phi_1^T \vee \neg\phi_1^T\widehat{\mathcal{S}}_{(0,r(J))} \ominus \phi_1^T \end{aligned}$$

- $\phi = \lhd_J \phi_1$ with $l(j) \neq 0$. And thus $r(J) = \infty$ as $\phi \in \mathsf{EventClockTL}_{0,\infty}$.
  By a similar reasoning we obtain:

$$\begin{aligned} \lhd_J\phi_1 = &\wedge \top\widehat{\mathcal{S}}_{(0,\infty)}\phi_1^T \\ &\wedge \neg\phi_1^T\widehat{\mathcal{S}}_{(0,\infty)}\phi_1^T \rightarrow \neg\phi_1^T\widehat{\mathcal{S}}_J\phi_1^T \\ &\wedge \neg(\neg\phi_1^T\widehat{\mathcal{S}}_{(0,\infty)}\phi_1^T) \rightarrow \neg\phi_1^T\widehat{\mathcal{S}}_{[l(J),\infty)}(\neg\phi_1^T \wedge \ominus\phi_1^T) \end{aligned}$$

□

We now prove that the reverse property also holds:

Lemma 3.34 ($\mathsf{MetricIntervalTL}_{0,\infty} \subseteq \mathsf{EventClockTL}$) *For every formula ϕ of $\mathsf{MetricIntervalTL}_{0,\infty}$, there exists a congruent formula ϕ^T of $\mathsf{EventClockTL}$, that is for every TSS κ and every time $t \in \mathbb{R}^+$: $(\kappa, t) \models \phi^T(t)$ iff $(\kappa, t) \models \phi$.*

Proof. We reason by induction on the structure of formulas. The interesting formulas are the $\widehat{\mathcal{U}}_I$ and $\widehat{\mathcal{S}}_I$ ones. In the sequel of the proof, we use the following usual abbreviations:

- $\widehat{\Diamond}_I\phi = \top\widehat{\mathcal{U}}_I\phi$;

- $\widehat{\Box}_I = \neg\widehat{\Diamond}_I\neg\phi$.

With the abbreviations given in definition 3.12, we can rewrite any $\widehat{\mathcal{U}}_I$-formulas as:

- if $l(I) = 0$ then $\phi_1\widehat{\mathcal{U}}_I\phi_2 = \phi_1\widehat{\mathcal{U}}_{(0,\infty)}\phi_2 \wedge \widehat{\Diamond}_I\phi_2$;
- if $I = (c,\infty)$ then $\phi_1\widehat{\mathcal{U}}_{(c,\infty)}\phi_2 = \widehat{\Box}_{(0,c]}(\phi_1 \wedge \phi_1\widehat{\mathcal{U}}_{(0,\infty)}\phi_2)$;
- if $I = [c,\infty)$ then $\phi_1\widehat{\mathcal{U}}_{[c,\infty)}\phi_2 = \begin{array}{l}\wedge\ \widehat{\Box}_{(0,c)}\phi_1 \\ \wedge\ \widehat{\Box}_{(0,c]}((\phi_1\widehat{\mathcal{U}}_{(0,\infty)}\phi_2) \vee \phi_2)\end{array}$;

Let us also note that :

- $\widehat{\Diamond}_{(c,\infty)}\phi = \widehat{\Box}_{(0,c]}\widehat{\Diamond}_{(0,\infty)}\phi$;
- $\widehat{\Diamond}_{[c,\infty)}\phi = \widehat{\Box}_{(0,c]}(\phi \vee \widehat{\Diamond}_{(0,\infty)}\phi)$;

So the only formula that we have to be able to treat are $\phi_1\widehat{\mathcal{U}}_{(0,\infty)}\phi_2$, $\widehat{\Diamond}_J\phi_1$ and $\widehat{\Box}_J\phi_1$ with $l(J) = 0$, and these are translated into EventClockTL as follows:

- $\phi_1\widehat{\mathcal{U}}_{(0,\infty)}\phi_2 = \phi_1^T\mathcal{U}(\phi_2^T \wedge \ominus\phi_1^T)$, where $\ominus\phi_1^T$ is the abbreviation for $\bot\mathcal{S}\phi_1^T$, see definition **??**;
- $\widehat{\Diamond}_J\phi_1 = \rhd_J\phi_1^T$;
- $\widehat{\Box}_J\phi_1 = \neg\rhd_J\neg\phi_1^T$.

The past temporal and real-time operators are treated symmetrically. This concludes our proof for $\mathsf{MetricIntervalTL}_{0,\infty} \subseteq \mathsf{EventClockTL}$. $\Box$

A direct consequence of the two previous lemmas is the following theorem:

Theorem 3.35 ($\mathsf{EventClockTL} = \mathsf{EventClockTL}_{0,\infty} = \mathsf{MetricIntervalTL}_{0,\infty}$) *The logics* EventClockTL, $\mathsf{EventClockTL}_{0,\infty}$ *and* $\mathsf{MetricIntervalTL}_{0,\infty}$ *are equally expressive.*

We now turn to the comparison of the expressive power of EventClockTL with regard to the expressive power of (full) MetricIntervalTL. A corollary of lemma 3.33 is that MetricIntervalTL is at least as expressive as EventClockTL. It could be thought that MetricIntervalTL has a strictly more expressive power than EventClockTL, but the following lemma and its proof, surprisingly, establishes that every MetricIntervalTL-formula is expressible in EventClockTL:

Lemma 3.36 ($\mathsf{MetricIntervalTL} \subseteq \mathsf{EventClockTL}$) *For every formula ϕ of* MetricIntervalTL*, there exists a congruent formula ϕ^T of* EventClockTL*, that is for every* TSS *κ and every time $t \in \mathbb{R}^+$: $(\kappa,t) \models \phi^T$ iff $(\kappa,t) \models \phi$.*

Proof. As we have proved in lemma 3.34 that $\mathsf{MetricIntervalTL}_{0,\infty} \subseteq \mathsf{EventClockTL}$, we are allowed to show that $\mathsf{MetricIntervalTL} \subseteq (\mathsf{EventClockTL} \cup \mathsf{MetricIntervalTL}_{0,\infty})$ and we have only to consider formulas that are not in $\mathsf{MetricIntervalTL}_{0,\infty}$. The interesting formulas of this fragment are of the form:

1. $\phi_1\widehat{\mathcal{U}}_I\phi_2$;
2. $\phi_1\widehat{\mathcal{S}}_I\phi_2$.

with $l(I) \neq 0$, $r(I) \neq \infty$ and I non-singular. In the following, we only consider the future formulas, past formulas are treated symmetrically. We first make a rewriting of those formulas to facilitate the rest of the proof:

- $\phi_1\widehat{\mathcal{U}}_I\phi_2$ with $l(I) \notin I$ can be rewritten as the following conjunction:

$1.\wedge\ \widehat{\Box}_{(0,l(I)]}(\phi_1 \wedge \phi_1 \widehat{\mathcal{U}}_{(0,\infty)} \phi_2)$
$2.\wedge\ \widehat{\Diamond}_I \phi_2$

- $\phi_1 \widehat{\mathcal{U}}_I \phi_2$ with $l(I) \in I$ can be rewritten as the following conjunction:

$1.\wedge\ \widehat{\Box}_{I \cap (0,l(I))} \phi_1$
$2.\wedge\ \widehat{\Box}_{(0,l(I)]}((\phi_1 \widehat{\mathcal{U}}_{(0,\infty)} \phi_2) \vee \phi_2)$
$3.\wedge\ \widehat{\Diamond}_I \phi_2$

And as each $\widehat{\Box}_I \phi$ formula can be rewritten as $\neg \widehat{\Diamond}_I \neg \phi$ formula, we have only to consider $\widehat{\Diamond}_I \phi$ formulas.

Let us now show that every formula $\widehat{\Diamond}_I \phi$ can be expressed in $\mathsf{EventClockTL} \cup \mathsf{MetricIntervalTL}_{0,\infty}$. We first rewrite those formulas as a disjunction of formulas where $l(I) = a$ and $r(I) = a + 1$. In fact, we have the following equivalence:

$$\widehat{\Diamond}_{(a,b]} \phi = \bigvee_{i=a}^{i=b-1} \widehat{\Diamond}_{(i,i+1]} \phi$$

This equivalence can be extended for all sorts of non-singular intervals (open-closed). We show in the sequel that each formula of the form $\widehat{\Diamond}_I \phi$, with $l(I) = c$ and $r(I) = c + 1$, can be expressed by an $\mathsf{EventClockTL} \cup \mathsf{MetricIntervalTL}_{0,\infty}$ formula and thus, by lemma 3.34 by an $\mathsf{EventClockTL}$ formula. The proof is by induction on the size of the constant c that appear in the constraining interval.

- *Base case.* When $c = 0$, the formula is in $\mathsf{MetricIntervalTL}_{0,\infty}$ and thus the base case is trivially verified.
- *Induction case.* We now treat the case for an arbitrary $c \in \mathbb{N}$. By induction hypothesis every formula of the form $\widehat{\Diamond}_I \phi$, with $l(I) \leq c - 1$ and $r(I) \leq c$ can be translated into $\mathsf{EventClockTL}$. We treat the case $\widehat{\Diamond}_{(c,c+1)} \phi$ in details, the other cases, i.e. $[c, c+1], [c, c+1), (c, c+1]$ are treated in the same way. Here is the translation:

$$\begin{aligned} \widehat{\Diamond}_{(c,c+1)} \phi = \ & a.\vee\ \widehat{\Diamond}_{[c-1,c)} \rhd_{=1} \bigcirc \phi \\ & b.\vee\ \widehat{\Diamond}_{(c-1,c)} \rhd_{=1} \phi \\ & c.\vee\ \widehat{\Box}_{(c-1,c]} \widehat{\Diamond}_{(0,1)} \phi \end{aligned}$$

We first prove that the implication from left to right is valid. There are two mutually exclusive situations to discriminate:

(1) In the first case, either the distance between the last ϕ-interval in $t + (c - 1, c]$ and the first ϕ-interval in $t + (c, c + 1)$ is greater or equal to 1 or there is no ϕ-interval in $t + [c - 1, c)$. We further distinguish two subcases:

(1a) the first ϕ_1 interval is left closed;

(1b) the first ϕ_1 interval is left open;

(2) In the second case, the distance between the last ϕ-interval in $t + (c - 1, c]$ and the first ϕ-interval in $t + (c, c + 1)$ is strictly less then 1.

In case 1: by the hypothesis that the distance between the first ϕ-interval in $t + (c, c+1)$, noted F_ϕ, and the last ϕ-interval in $t + (c - 1, c]$, noted L_ϕ is greater than 1, we infer that there exists $t_1 \in t + (c - 1, c]$ such that $(\kappa, t_1) \models \rhd_{=1} \phi$ if the interval I_ϕ is left-closed (1a) and that $(\kappa, t_1) \models \rhd_{=1} \bigcirc \phi$ if I_ϕ is left open (1b). Using the induction hypothesis, we express this property with the $\widehat{\Diamond}_{(c-1,c)}$ operator in the first case and with the $\widehat{\Diamond}_{[c-1,c)}$ operator in the second case.

In case 2: the distance between the last ϕ-interval in $t + (c - 1, c]$ noted L_ϕ and the first ϕ-interval in $t + (c, c + 1)$, noted F_ϕ is strictly less than one. For all time $t \in (c - 1, r(L_\phi))$, $\widehat{\Diamond}_{(0,1)} \phi$ is verified thanks to ϕ-positions in L_ϕ and for all time $t \in [r(L_\phi), c]$, $\widehat{\Diamond}_{(0,1)}\phi$ is verified thanks to ϕ-positions in F_ϕ (as the distance is less than 1).

The other direction is immediate. We must show that the three parts of the disjunction implies the MetricIntervalTL formula $\widehat{\Diamond}_{(c,c+1)}\phi$:

1. $\widehat{\Diamond}_{[c-1,c)} \rhd_{=1} \bigcirc\phi$. Clearly this formula asserts that there is a time $t_1 \in t + [c, c+1)$ such that at a distance of 1 time unit $\bigcirc\phi$ is verified, let us note this position $t_2 = t_1 + c$. So there is a left-open ϕ-interval at a distance of $1 + [c-1, c)$ from t and thus as this ϕ-interval is left-open, we have that $\widehat{\Diamond}_{(c,c+1)}\phi$ is verified in time t.
2. $\widehat{\Diamond}_{(c-1,c)} \rhd_{=1} \phi$. By the same reasoning but for a left-closed interval, we establish that $\widehat{\Diamond}_{(c,c+1)}\phi$ is verified in time t;
3. $\widehat{\Box}_{(c-1,c]}\widehat{\Diamond}_{(0,1)}\phi$. This formula directly implies that $\widehat{\Diamond}_{(0,1)}\phi$ is verified in time $t + c$. So there is a time $t_1 \in t + c + (0,1)$ where ϕ is verified as $t_1 \in t + (c, c+1)$ we have that $\widehat{\Diamond}_{(c,c+1)}\phi$ is verified at time t.

The equivalence between the two formula is proved. As the formula $\widehat{\Box}_{(c-1,c]}\phi$ is equivalent to the formula $\neg\widehat{\Diamond}_{(c-1,c]}\neg\phi$ and that the constant appearing in the left-end bound of the constraining interval is strictly less than c, by induction hypothesis, the formula $\neg\widehat{\Diamond}_{(c-1,c]}\neg\phi$ can be expressed in EventClockTL.

□

The last lemma together with the lemma 3.33 gives:

Theorem 3.37 (EventClockTL = MetricIntervalTL) *The logics* EventClockTL *and* MetricIntervalTL *are equally expressive.*

Corollary 3.38 (All Equally Expressive) *The logics* EventClockTL, $\mathsf{EventClockTL}_{0,\infty}$, $\mathsf{MetricIntervalTL}_{0,\infty}$, MetricIntervalTL *and* $\mathsf{MinMaxML}_1$ *are equally expressive.*

That is, all the logics define the same class of real-time ω-languages. We call this class the *counter-free ω-regular real-time languages.*

Definition 3.39 (Class of ω-Regular Real-Time Languages) The sets of timed state sequences definable by the logics EventClockTL, $\mathsf{EventClockTL}_{0,\infty}$, $\mathsf{MetricIntervalTL}_{0,\infty}$, MetricIntervalTL and $\mathsf{MinMaxML}_1$ form the class of *counter-free ω-regular real-time languages.*

3.5.3 Minimal Expressively Complete Fragments

In this section, we identify minimal fragments that are fully expressive. We show that in each of the previously defined logics, we can restrict the use of constants to be only 0 or 1.

Definition 3.40 ($\mathsf{MetricIntervalTL}_{0,1}$-Fragment) $\mathsf{MetricIntervalTL}_{0,1}$ is the fragment of MetricIntervalTL that consists of all formulas ϕ such that for each interval I appearing in ϕ, we have $l(I) = 0$ and $r(I) = 1$. □

Similarly,

Definition 3.41 ($\mathsf{EventClockTL}_{0,1}$-Fragment) $\mathsf{EventClockTL}_{0,1}$ is the fragment of EventClockTL that consists of all formulas ϕ such that for each interval I appearing in ϕ, we have $l(I) = 0$ and $r(I) = 1$. □

We have the following lemma:

Lemma 3.42 ($\mathsf{MetricIntervalTL}_{0,\infty} \subseteq \mathsf{MetricIntervalTL}_{0,1}$) *For every formula ϕ of* $\mathsf{MetricIntervalTL}_{0,\infty}$, *there exists a congruent formula ϕ^T of* $\mathsf{MetricIntervalTL}_{0,1}$, *that is for every* TSS *κ and every time $t \in \mathbb{R}^+$: $(\kappa, t) \models \phi^T$ iff $(\kappa, t) \models \phi$.*

Proof. In the proof of lemma 3.34, we have shown that every $\mathsf{MetricIntervalTL}_{0,\infty}$ formula can be rewritten using only the following real-time formulas: $\phi_1 \widehat{\mathcal{U}}_{(0,\infty)} \phi_2$ and $\widehat{\Diamond}_J \phi_1$ with $l(J) = 0$. So all we need to consider is formulas of the form $\Diamond_{<c}\phi_1$, $\Diamond_{\leq c}\phi_1$. We treat the case $\Diamond_{<c}\phi_1$, the other cases are treated similarly and left to the reader. We reason by induction on the size of the constant c and make the hypothesis that we can effectively construct the formula ϕ_1^T.

- *Base case*: $c = 1$. Then $\Diamond_{<1}\phi_1^T$ is already in $\mathsf{MetricIntervalTL}_{0,1}$.
- *Induction case*: $c > 1$ and by induction hypothesis we can handle formulas $\Diamond_{<d}\phi_1^T$, with $0 \leq d < c$. For $\Diamond_{<c}\phi_1$, we take: $\Diamond_{<1}(\Diamond_{c-1}\phi_1)^T$, which by induction hypothesis, is in $\mathsf{MetricIntervalTL}_{0,1}$.

□

As a consequence of this lemma and corollary 3.38, we have the following corollary:

Corollary 3.43 *The logics* $\mathsf{MetricIntervalTL}$ *and* $\mathsf{MetricIntervalTL}_{0,1}$ *are equally expressive.*

Lemma 3.44 ($\mathsf{EventClockTL} \subseteq \mathsf{EventClockTL}_{0,1}$) *For every formula ϕ of* $\mathsf{EventClockTL}$*, there exists a congruent formula ϕ^T of* $\mathsf{EventClockTL}_{0,1}$*, that is for every* TSS *κ and every time $t \in \mathbb{R}^+$: $(\kappa, t) \models \phi^T$ iff $(\kappa, t) \models \phi$.*

Proof. In lemma 3.32, we have shown that $\mathsf{EventClockTL} \subseteq \mathsf{EventClockTL}_{0,\infty}$. Thus, we must show that $\rhd_{\sim c}\phi_1$ with $\sim \in \{<, \leq, \geq, >\}$ can be translated into $\mathsf{EventClockTL}_{0,1}$. We treat $\rhd_{\leq c}\phi_1$ and $\rhd_{\geq c}\phi_1$, the other cases are similar and left to the reader.

- $\phi = \rhd_{\leq c}\phi_1$. We reason by induction on the size of c.
  - $c = 1$. In that case $\rhd_{\leq 1}\phi_1$ is an $\mathsf{EventClockTL}_{0,1}$ formula.
  - $c > 1$. By induction hypothesis, we can treat every formula of $\mathsf{EventClockTL}_{0,\infty}$ with a constant $d < c$. Then we take $\rhd_{\leq c}\phi_1 = \rhd_{\leq 1}(\rhd_{\leq c-1}\phi_1)^T$.
- $\phi = \rhd_{\geq c}\phi_1$. Note that we can rewrite this formula as follows: $\neg(\rhd_{<c}\phi_1) \wedge \Diamond\phi_1$. By the previous case, we know that we can transform $\rhd_{<c}\phi_1$ into an $\mathsf{EventClockTL}_{0,1}$ formula.

□

A direct consequent of the previous lemma and corollary 3.38, we have the following corollary:

Corollary 3.45 *The logics* $\mathsf{MetricIntervalTL}$*,* $\mathsf{MetricIntervalTL}_{0,\infty}$*,* $\mathsf{MetricIntervalTL}_{0,1}$*,* $\mathsf{EventClockTL}$*,* $\mathsf{EventClockTL}_{0,1}$ *and* $\mathsf{MinMaxML}_1$ *are equally expressive.*

4 The Regular Real-Time ω-Languages

4.1 Introduction

In this section, we will study automata that are closely related to the logic of event clocks. This class of automata, called the recursive event-clock automata is study in details: we study its closure properties, decidability results as well as expressiveness results. It will turn out that the class of language accepted by the recursive event-clock automata is exactly the languages accepted by the logics of the previous section when ability to count is added. For this reason, we call the languages accepted by the recursive event-clock automata the "(full) regular real-time languages". This class of languages is closed under all boolean operations.

4.2 Propositional Event-Clock Automata

An event-clock automaton is a special case of a timed automaton [AD94], where the starting of clocks is determined by the input instead of by the transition relation. We first recall the original definition with event clocks associated to proposition [AFH94].

The value of propositional event clocks in the continuous semantics will be non standard reals, which are defined as follows:

Definition 4.1 (Non-Standard Reals) The set of *non-standard (positive) reals*, noted $\mathbb{R}^+_{ns}$, is the set $\{v, v^+ \mid v \in \mathbb{R}^+\}$, ordered by $<_{ns}$ as follows: $v_1 <_{ns} v_2^+$ iff $v_1 \leq v_2$ where $\leq$ is the usual order on real-numbers. $\square$

We are now equipped to define the value of propositional event clocks along timed state sequences.

Definition 4.2 (Value of Event Clocks-Continuous Semantics) The *value of an propositional event clock* $z \in \mathbb{C}$ along a TSS κ, at time t, noted $\mathsf{Val}_\kappa(z,t)$ is defined by the following clauses:

$$\mathsf{Val}_\kappa(x_p,t) = \begin{cases} v & \text{if } p \in \kappa(t-v),\ v > 0, \\ & \quad \text{and for all } v',\, 0 < v' < v, \text{ not } p \in \kappa(t-v') \\ v^+ & \text{if for all } v' > v, \text{ exists } v'',\, v < v'' < v' \text{ with } p \in \kappa(t-v''), \\ & \quad \text{and for all } v',\, 0 < v' \leq v, \text{ not } p \in \kappa(t-v') \\ \perp & \text{if for all } v,\, 0 < v \leq t, \text{ not } p \in \kappa(t-v) \end{cases}$$

$$\mathsf{Val}_\kappa(y_p,t) = \begin{cases} v & \text{if } p \in \kappa(t+v),\ v > 0, \\ & \quad \text{and for all } v',\, 0 < v' < v, \text{ not } p \in \kappa(t+v') \\ v^+ & \text{if for all } v' > v, \text{ exists } v'',\, v < v'' < v' \text{ with } p \in \kappa(t+v''), \\ & \quad \text{and for all } v',\, 0 < v' \leq v, \text{ not } p \in (t+v') \\ \perp & \text{if for all } v > 0, \text{ not } p \in \kappa(t+v) \end{cases}$$

$\square$

Definition 4.3 (Atomic Event Clock Constraints) Given a set of (propositional) event clocks $\mathbb{C}$, the set of atomic clock constraints is $\{z \sim c \mid z \in \mathbb{C} \text{ and } c \in \mathbb{N}\}$. $\square$

Let us now show how the truth value of atomic event clock constraints is evaluated along a TSS:

Definition 4.4 (Clock Constraints Semantics) A atomic event clock constraints $z \sim c$ is true at time $t \in \mathbb{R}^+$ of the TSS κ, noted $(\kappa, t) \models z \sim c$, iff $\mathsf{Val}_\kappa(z,t) \sim c$. $\square$

Definition 4.5 (Propositional Event-Clock Automata) A *propositional event-clock automaton*, in the continuous semantics, is a tuple $A = (Q, Q_0, \delta, \mathcal{P}, \mathcal{A}, \lambda, Q_F)$ where:

Q is a finite set of locations,
$Q_0 \subseteq Q$ is the set of starting locations,
$\delta \subseteq Q \times Q$ is the transition relation,
$\mathcal{P}$ is a finite set of propositional symbols,
$\mathcal{A}$ is a finite set of atomic real-time constraints over propositional clocks,
λ: $Q \to 2^{\mathsf{Limit}(\mathcal{P} \cup \mathcal{A})}$ is a function that labels each location with a set of literals;
$Q_F \subseteq Q$ is a set of accepting locations.

$\square$

Let us note that we label here the locations with set of literals. We could have decided to label locations with boolean combinations of literals instead. We just adopt this convention because it will slightly simplify some proofs later but the expressive power would have been the same if we had chosen to label with boolean combinations of literals instead. We now define formally the notion of accepted timed run of a $\mathsf{EventClockTA}$ on a TSS κ. Let κ be a timed state sequence whose propositional symbols contain all propositions in $\mathcal{P}$.

Definition 4.6 (Accepted Timed Run) The propositional event-clock automaton A *accepts* κ, denoted $\mathsf{Accept}_A(\kappa)$, iff there exist an accepted infinite timed run $\rho = (\overline{q}, \overline{I})$ such that the following conditions are met.

Covering The run ρ consists of an infinite sequence $\overline{q}$ of locations from Q, and an infinite interval sequence $\overline{I}$ that covers $[0, \infty)$.

Starting The run starts in a starting location, i.e. $q_0 \in Q_0$.

Consecution The run respects the transition relation; that is, $(q_i, q_{i+1}) \in \delta$ for all $i \geq 0$.

Constraints The timed state sequence respects the constraints that are induced by the run ρ; that is, $\kappa(t) \models \lambda(\rho(t))$ for all real times $t \in [0, \infty)$.

Accepting The run is Büchi accepting, that is, there exist infinitely many $i \geq 0$ such that $q_i \in Q_F$.

□

Each EventClockTA defines a real-time ω-regular language:

Definition 4.7 (Continuous Anchored Real-Time Language) The *continuous anchored real-time language* defined by an propositional event-clock automaton A, noted $\mathsf{AncLang}(A)$ is the set of TSS on which it has an accepted run, that is $\mathsf{AncLang}(A) = \{\kappa \mid \mathsf{Accept}_A(\kappa)\}$.

Theorem 4.8 (Closure Properties) *The formalism of propositional event-clock automaton is (constructively) closed, in the continuous semantics, under all boolean operations.* □

By slightly adapting the region construction presented in section **??**, we can also construct, for each EventClockTAA, a BA R^A that accepts the untimed A:

Theorem 4.9 (Region Automaton) *For every (continuous) propositional event clock A, we can construct a Büchi automaton B with $\mathsf{AncLang}(B) = \{\overline{\sigma} \mid (\overline{\sigma}, \overline{I}) \in \mathsf{AncLang}(A)\}$. Further the number of locations in B is linear in the number of locations used in A, singly exponential in the number of clocks used in A and singly exponential in the size of the maximal constant used in A.* □

The last theorem and the closure properties of continuous propositional event clock automata allow us to derive:

Theorem 4.10 (Emptiness and Universality of EventClockTA) *The emptiness and universality problems for (propositional) event clock automata in continuous semantics are decidable and* PSPACE-COMPLETE.
□

Unfortunately, the propositional version of event-clock automata does not subsume the logic EventClockTL.

Theorem 4.11 (EventClockTL $\not\subseteq$ EventClockTA, EventClockTA $\not\subseteq$ EventClockTL) *The expressive power of continuous* EventClockTL *and continuous* EventClockTA *are incomparable.*

Proof. The non inclusion of the EventClockTA-languages in the EventClockTL-languages is as for the pointwise case: the logic EventClockTL is not able to express counting properties. For the non inclusion of the EventClockTL-languages in the EventClockTA-languages, we consider the two TSS $\kappa^1 = (\overline{\sigma}, \overline{I}^1)$ and $\kappa^2 = (\overline{\sigma}, \overline{I}^2)$ defined on the singleton $\{p\}$:

- the two TSS share the same qualitative information which is as follows: $\overline{\sigma} = \{\}\{\}\{p\}\{\}\{p\}\{p\}\{p\}\ldots$, that is p is false in the two first observations, becomes true in the third observation, becomes false again in the fourth observation and then true for ever from the fifth observation.
- let us now consider the two following sequences of intervals:

1. $\overline{I}^1 = [0,0](0.5;0.5)[0.5;0.5](0.5;1)[1;1](1;1.5)[1.5;1.5]\ldots$, that is every interval I_i^1 with i even is singular and equal to $[(i-1)\times 0.5;(i-1)\times 0.5]$;
2. $\overline{I}^2 = [0,0](0.4;0.4)[0.4;0.4](0.4;0.8)[0.8;0.8](0.8;1.2)[1.2;1.2]\ldots$, that is every interval I_i^2 with i even is singular and equal to $[(i-1)\times 0.4;(i-1)\times 0.4]$;

It is easy to show that for every clock constraint $z \sim c$ that we can build from the propositional prophecy clocks x_p and y_p, that we have the following property: for every positions $i \geq 0$, for every $t_1^1, t_2^1 \in I_1^1$, for every $t_1^2, t_2^2 \in I_1^1$, we have that: $(\kappa^1, t_1^1) \models z \sim c$ iff $(\kappa^1, t_2^1) \models z \sim c$ iff $(\kappa^2, t_1^2) \models z \sim c$ iff $(\kappa^2, t_2^2) \models z \sim c$. As the two timed state sequences are alternating, we have the same property for every atom build from propositions and atomic clock constraints. And thus every EventClockTA either accepts or rejects the two TSS. On the order hand, the EventClockTL formula $\phi = \rhd_{=1}\Box p$ is true in time $t = 0$ of the first TSS but false in $t = 0$ of the second. As a consequence, no EventClockTA can express the property expressed by the EventClockTL formula ϕ.

□

This result motivates the following extension. We extend the use of event clocks: propositional event clocks are clocks that can only be associated to propositional symbols, here we show that we can associate event clocks with automata recursively. The formalism that we obtain is called the *recursive* event-clock automata. Those recursive automata keep all nice properties of their propositional version: closure under *all boolean operations* and both *emptiness* and *universality* problems are decidable. Further, we will show that contrary to propositional event-clock automata, recursive event-clock automata are able to express all EventClockTL-expressible properties.

4.3 Recursive Event-Clock Automata: REventClockTA

We now generalize the use of clocks to define our recursive event-clock automata, noted REventClockTA. An automaton A accepts (or rejects) a given pair (κ, t) that consists of a timed state sequence κ and a time $t \in \mathbb{R}^+$. The automaton is started at time t and views the "past" of the input sequence κ by executing a *backward* transition relation, and the "future" by executing to a *forward* transition relation. If A accepts the pair (κ, t), we say that A accepts κ at time t. This allows us to associate a history clock and a prophecy clock with each automaton. The history clock x_A always shows the amount of time that has expired since the last time at which A accepted κ, and the prophecy clock y_A always shows the amount of time that will expire until the next time at which A will accept κ. This definition of event-clock automata is recursive. The base automata, whose transition relations are not constrained by clocks, are called *floating automata*, FloatA for short. Formally,

Definition 4.12 (FloatA) A *floating automaton* is a tuple $A = (Q, Q_0, \delta_f, \delta_b, \mathcal{P}, \lambda, Q_{F_f}, Q_{F_b})$ such that

Q is a finite set of locations,
$Q_0 \subseteq Q$ is the set of starting locations,
$\delta_f \subseteq Q \times Q$ is the forward transition relation,
$\delta_b \subseteq Q \times Q$ is the backward transition relation,
$\mathcal{P}$ is a finite set of propositional symbols,
λ: $Q \to 2^{\mathsf{Limit}(\mathcal{P})}$ is a function that labels each location with a set of literals over the set of propositions $\mathcal{P}$;
$Q_{F_f} \subseteq Q$ is a set of forward accepting locations, and
$Q_{F_b} \subseteq Q$ is a set of backward accepting locations.

□

Note that we have chosen to label locations with the set of literals that are true when the control reside in the location. We have done this choice because it will slightly simplify some proofs later. But for specification convenience, we could have chosen, with no effect on the property of our recursive event-clock automata, to label locations with boolean formulas built from those literals. We will use those boolean formulas when illustrating the use of recursive event-clock automata for specifying real-time properties. Examples will be more readable with this convention.

We now define the notion of accepted timed run for floating automata on a pair (κ, t).

Definition 4.13 (FloatA-Accepted Run) Let κ be a timed state sequence whose propositional symbols contain all propositions in $\mathcal{P}$. The floating automaton A *accepts* κ at time $t \in \mathbb{R}^+$, denoted $\mathsf{Accept}_A(\kappa, t)$, iff there exist an infinite forward timed run $\rho^f = (\overline{q}^f, \overline{I}^f)$ and a finite backward timed run $\rho^b = (\overline{q}^b, \overline{I}^b)$ such that the following conditions are met. We note $\rho(t)$ the location in which the run resides at time $t \in \mathbb{R}^+$.

Covering The forward run ρ^f consists of an infinite sequence $\overline{q}^f$ of locations from Q, and an infinite interval sequence $\overline{I}$ that covers $[t, \infty)$. The backward run ρ^b consists of a finite sequence $\overline{q}^b$ of locations and a finite interval sequence $\overline{I}^b$, of the same length as $\overline{q}^b$, which covers $[0, t]$.

Starting The forward and backward runs start in the same starting location; that is, $\rho^f(t) = \rho^b(t)$ and $\rho^f(t) \in Q_0$.

Consecution The forward and backward runs respect the corresponding transition relations; that is, $(q_i^f, q_{i+1}^f) \in \delta_f$ or $q_i^f = q_{i+1}^f$ (stuttering) for all $i \geq 0$, and $(q_i^b, q_{i-1}^b) \in \delta^b$ or $q_i^b = q_{i-1}^b$ (stuttering) for all $0 < i < |\overline{q}^b|$.

Constraints The timed state sequence respects the constraints that are induced by the forward and backward runs; that is, $(\kappa, t') \models \lambda(\rho^f(t'))$ for all real times $t' \in [t, \infty)$, and $(\kappa, t') \models \lambda(\rho^b(t'))$ for all real times $t' \in [0, t]$.

Accepting The forward run is Büchi accepting and the backward run ends in an backward accepting location; that is, there exist infinitely many $i \geq 0$ such that $q_i^f \in Q_{F_f}$, and $q_0^b \in Q_{F_b}$.

□

Example 4.14 (A Floating Automaton) The simple floating automaton A of figure 2 has the following elements:

- location q_1 is the starting location of A;
- its forward transition relation (plain arrows) allows the control to evolve from location q_1 to location q_2 and to loop in location q_2;
- its backward transition relation (dashed arrows) allows the control to reach location q_0 from location q_1 and afterwards to loop in location q_0;
- location q_0 is backward accepting (double circle dashed) and location q_2 is forward accepting (double circle);
- its labels are as follows: in location q_1, the literal $\overrightarrow{p}$ must hold, it means that p must be true just after the time at which the automaton is started (but not necessary at the time the automaton is started); when the control resides in location q_2, the proposition p must be true; when the control is in location q_1, no constraint are imposed (the location is labeled with the literal $\top$).

Following the rules given in definition 4.13, it is not difficult to see the floating automaton A accepts exactly the pairs (κ, t) such that p is always true just after t, that is the pairs where the $\mathsf{EventClockTL}$ formula $\Box p$ evaluates positively, i.e. $(\kappa, t) \models \Box p$.

We are now in position to define the notion of recursive automaton of level i:

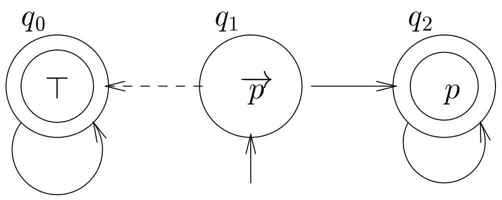

Figure 1: Floating Automaton A.

Definition 4.15 (REventClockTA) A *recursive event-clock automaton of level* $i \in \mathbb{N}$ is a tuple $A = (Q, Q_0, \delta_f, \delta_b, \mathcal{P}, \mathcal{A}, \lambda, Q_{F_f}, Q_{F_b})$ that has the same components as a floating automaton plus a set of atomic clock constraints $\mathcal{A}$ over the set of level_i clocks, noted Γ_i, that can be used by the labeling function λ: $Q \to 2^{\mathsf{Limit}(\mathcal{P}\cup\mathcal{A})}$ that labels each location with a set of literals over propositions and level_i clock constraints. The set Γ_i of *level-i clock constraints* contains all atomic formulas of the form $x_B \sim c$ and $y_B \sim c$, where B is a recursive event-clock automaton of level less than i whose propositions are contained in $\mathcal{P}$, where c is a nonnegative integer constant, and where $\sim \in \{<, \leq, =, \geq, >\}$. The clock x_B is called the *history clock* of automaton B, and the clock y_B, the *prophecy clock* of automaton B. $\Box$

In particular, the set of level_0 clock constraints is empty, and thus the level_0 event-clock automata are the floating automata. The level_1 clock constraints are the clock constraints built using event clocks associated with floating automata...

Definition 4.16 (Subautomata) If A contains a constraint on x_B or y_B, we say that B is a *subautomaton* of A. We use the notation $\mathsf{SUB}(A)$ to denote the set of subautomata used in A or recursively, in a subautomaton of A.

The definition of when the recursive event-clock automaton A of level i accepts a timed state sequence κ at time t is as for floating automata, only that we need to define the satisfaction relation $(\kappa, t) \models (z \sim c)$ for every time $t \in \mathbb{R}^+$ and every level_i clock constraint $(z \sim c) \in \Gamma_i$. The rules for evaluating the truth value of a clock constraint are as in the propositional case. We only need to define the value of recursive event clocks. This is done as follows.

Definition 4.17 (Recursive Event-Clock Value) The *value of a recursive event-clock* $z_B \in \mathbb{C}$ with $\mathsf{level}(z_A) = 1$ along a TSS κ, at time t, noted $\mathsf{Val}_\kappa(z_A, t)$ is defined by the following clauses:

$$\mathsf{Val}_\kappa(x_B, t) = \begin{cases} v & \text{if } \mathsf{Accept}_B(\kappa, t-v),\ v > 0, \\ & \quad \text{and for all } v',\ 0 < v' < v, \text{ not } \mathsf{Accept}_B(\kappa, t-v') \\ v^+ & \text{if for all } v' > v, \text{ exists } v'',\ v < v'' < v' \text{ with } \mathsf{Accept}_B(\kappa, t-v''), \\ & \quad \text{and for all } v',\ 0 < v' \leq v, \text{ not } \mathsf{Accept}_B(\kappa, t-v') \\ \bot & \text{if for all } v,\ 0 < v \leq t, \text{ not } \mathsf{Accept}_B(\kappa, t-v) \end{cases}$$

$$\mathsf{Val}_\kappa(y_B, t) = \begin{cases} v & \text{if } \mathsf{Accept}_B(\kappa, t+v),\ v > 0, \\ & \quad \text{and for all } v',\ 0 < v' < v, \text{ not } \mathsf{Accept}_B(\kappa, t+v') \\ v^+ & \text{if for all } v' > v, \text{ exists } v'',\ v < v'' < v' \text{ with } \mathsf{Accept}_B(\kappa, t+v''), \\ & \quad \text{and for all } v',\ 0 < v' \leq v, \text{ not } \mathsf{Accept}_B(\kappa, t+v') \\ \bot & \text{if for all } v > 0, \text{ not } \mathsf{Accept}_B(\kappa, t+v) \end{cases}$$

where $\mathsf{Accept}_B(\kappa, t)$ is as in definition 4.13. The recursive case is treated as follows. By induction hypothesis, $\mathsf{Accept}_B(\kappa, t)$ is defined for every automaton B of level_j, with $0 \leq j < i$, the value of recursive clock of level i is simply:

$$\mathsf{Val}_\kappa(x_B, t) = \begin{cases} v & \text{if } \mathsf{Accept}_B(\kappa, t-v),\ v > 0, \\ & \quad \text{and for all } v',\ 0 < v' < v, \text{ not } \mathsf{Accept}_B(\kappa, t-v') \\ v^+ & \text{if for all } v' > v, \text{ exists } v'',\ v < v'' < v' \text{ with } \mathsf{Accept}_B(\kappa, t-v''), \\ & \quad \text{and for all } v',\ 0 < v' \leq v, \text{ not } \mathsf{Accept}_B(\kappa, t-v') \\ \bot & \text{if for all } v,\ 0 < v \leq t, \text{ not } \mathsf{Accept}_B(\kappa, t-v) \end{cases}$$

$$\mathsf{Val}_\kappa(y_B, t) = \begin{cases} v & \text{if } \mathsf{Accept}_B(\kappa, t+v),\ v > 0, \\ & \quad \text{and for all } v',\ 0 < v' < v, \text{ not } \mathsf{Accept}_B(\kappa, t+v') \\ v^+ & \text{if for all } v' > v, \text{ exists } v'',\ v < v'' < v' \text{ with } \mathsf{Accept}_B(\kappa, t+v''), \\ & \quad \text{and for all } v',\ 0 < v' \leq v, \text{ not } \mathsf{Accept}_B(\kappa, t+v') \\ \bot & \text{if for all } v > 0, \text{ not } \mathsf{Accept}_B(\kappa, t+v) \end{cases}$$

□

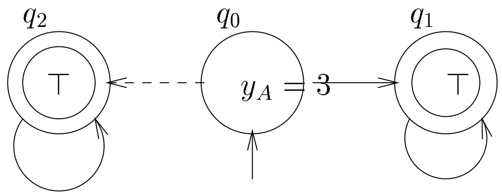

Figure 2: Recursive Event-Clock Automaton B.

Example 4.18 (A Recursive Event-Clock Automaton) Let us consider $\mathsf{MEventClockTA}$ of figure 2. As q_0 is the starting location of B, if B accepts (κ, t) then the first following time that A is accepting κ after time t is at time $t+3$. So the automaton B expresses exactly the semantics of the formula $\rhd_{=3}\Box p$.

For our recursive event-clock automata, we define two notions of timed languages: the *anchored language* and the *floating language*. The anchored real-time language is the traditional notion when the floating real-time language capture the notion of floating acceptance. The two types of real-time languages are defined formally as follows:

Definition 4.19 ($\mathsf{REventClockTA}$-Languages) A recursive event-clock automaton A defines the *floating timed language* $\{(\kappa, t) \mid \mathsf{Accept}_A(\kappa, t)\}$, that is, the floating language of A is the set of pairs (κ, t) that it accepts; we note $\mathsf{FloatLang}(A)$ the floating real-time language defined by A. Furthermore, A defines the *anchored language* $\{\kappa \mid \mathsf{Accept}_A(\kappa, 0)\}$ that is the set of TSS that A accepts at time 0; we note $\mathsf{AncLang}(A)$ the anchored language defined by A. □

The expressive power of recursive event-clock automata will be measured in term of its ability to define anchored real-time languages but the floating real-time languages are important in the proofs.

In what follows, we use two notions of equivalence for automata:

Definition 4.20 (Equivalent and Congruent Automata) Two recursive event-clock automata are *equivalent* if they define the same anchored language and they are *congruent* if they define the same floating language. □

Let us note that the notion of congruence is stronger than the notion of equivalence, that is: two congruent automata are equivalent but two equivalent automata are not necessarily congruent.

In the proofs of the following section, we will need the following notion. As for timed state sequences, we define a notion of refinement for forward and backward timed runs:

Definition 4.21 (Run Refinement) A forward (resp. backward) timed run $\rho^2 = (\overline{q}^2, \overline{I}^2)$ is a refinement of a forward (resp. backward) timed run $\rho^1 = (\overline{q}^1, \overline{I}^1)$ iff there exists a surjective function $f : \mathbb{N} \to \mathbb{N}$ such that:

- for all positions j with $0 \leq j < |\rho^2|$, $q^2_j = q^1_{f(j)}$;
- for all positions $0 \leq i < |\rho^1|$, $I^1_i = \bigcup\{I^2_j \mid f(j) = i\}$

where $|\rho|$ denotes the length of ρ, which is a finite natural number in the case of a backward timed run and ∞ in the case of a forward timed run. □

In what follows, we need the following lemma, which is a direct consequence of the possibility to take stuttering steps in timed runs:

Lemma 4.22 (Run Refinable) *If ρ^{f_1} and ρ^{b_1} are accepted forward and backward timed runs of A on the* TSS *κ at time $t \in \mathbb{R}^+$ then all forward and backward timed runs ρ^{f_2}, ρ^{b_2} such that ρ^{f_2} refines ρ^{f_1} and ρ^{b_2} refines ρ^{b_1}, are timed accepted runs of A on the* TSS *κ at time $t \in \mathbb{R}^+$.* □

4.4 Closure Properties of Recursive Event-Clock Automata

We now analyze the properties of our recursive event-clock automata. In order to enhance the readability of the proofs, we first define a variant of the definition of recursive event-clock automata given above. We call this variant "*monitored recursive event-clock automata*", noted MEventClockTA. In those automata, the forward and backward transition relations are replaced by a unique transition relation and the notion of floating acceptance is handled with a set of locations that we call *monitored.* We define formally the monitored event-clock automata and prove that their expressive power in term of anchored as well as floating languages, is equal to the expressive power of recursive event-clock automata. Again, we first define the base case.

Definition 4.23 (Monitored Floating Automata) A *monitored floating automaton* is a tuple $A = (Q, Q_0, Q_M, \delta, \mathcal{P}, \lambda, Q_F)$ where:

- Q is a finite set of locations,
- $Q_0 \subseteq Q$ is the set of starting locations,
- $Q_M \subseteq Q$ is the set of monitored locations,
- $\delta \subseteq Q \times Q$ is the transition relation,
- $\mathcal{P}$ is a finite set of propositional symbols,
- λ: $Q \to 2^{\mathsf{Limit}(\mathcal{P})}$ is a function that labels each location with a set of literals over propositions;
- $Q_F \subseteq Q$ is a set of accepting locations (Büchi condition).

□

We now define when an monitored floating automaton accepts a timed state sequence κ at time t.

Definition 4.24 (Monitored Timed Run) A *monitored floating automaton* $A = (Q, Q_0, Q_M, \delta, \mathcal{P}, \lambda, Q_F)$ accepts the the timed state sequence κ at time t, noted $\mathsf{Accept}_A(\kappa, t)$ iff there exists a timed run $\rho = (q_0, I_0), (q_1, I_1), \ldots, (q_n, I_n), \ldots$ such that:

Covering The run ρ cover the entire real time line, i.e. $\cup_i I_i = [0, \infty)$;

Starting The run ρ starts in a starting location of A, that is $q_0 \in Q_0$;

Monitoring The run ρ is in a monitored location at time t, i.e. $\rho(t) \in Q_M$;

Consecution The run ρ respects the transition relation of A, i.e. for all i such that $1 \leq i$, we have $(q_i, q_{i+1}) \in \delta$ or $q_i = q_{i+1}$;

Constraints The TSS κ respects the constraints induced by the timed run; that is for all time $t \in [0, \infty)$ we have that $(\kappa, t) \models \lambda(\rho(t))$;

Accepting The run ρ has infinitely many positions in the set of accepting locations, that is there exists infinitely many $i \geq 0$ such that $q_i \in Q_F$ (Büchi acceptance condition).

We call such run ρ an *t-monitored and accepted* run of A on κ, noted $\mathsf{Accept}_A(\kappa, t)$. □

We call the monitored floating automata, level_0 monitored recursive event-clock automata. A recursive monitored event-clock automaton of level_i has the ability to use clock associated recursively to automata of level_j, with $0 \leq j < i$. Formally,

Definition 4.25 (Monitored Recursive Automata) A *monitored recursive event-clock automaton* of level i is a tuple $A = (Q, Q_0, Q_M, \delta, \mathcal{P}, \mathcal{A}, \lambda, Q_F)$ such that

Q is a finite set of locations,
$Q_0 \subseteq Q$ is the set of starting locations,
$Q_M \subseteq Q$ is the set of monitored locations,
$\delta \subseteq Q \times Q$ is the transition relation,
$\mathcal{P}$ is a finite set of propositional symbols,
$\mathcal{A}$ is a finite set of atomic clock constraints over clocks of at most level i,
λ: $Q \to 2^{\mathsf{Limit}(\mathcal{P} \cup \mathcal{A})}$ is a function that labels each location with a set of literals over propositions and level-i clock constraints;
$Q_F \subseteq Q$ is a set of accepting locations (Büchi condition).

□

The definition of when a recursive monitored automata accepts a TSS κ at a given time $t \in \mathbb{R}^+$ is as expected. We now show that the variant that we have defined is exactly as expressive as the recursive event-clock automata for defining floating languages (and thus also anchored languages):

Lemma 4.26 ($\mathsf{REventClockTA} \subseteq \mathsf{MEventClockTA}$) *For every recursive event-clock automata A, we can construct a monitored recursive event-clock automata B that accepts exactly the same floating language.*

Proof. Our proof is constructive. We define a function T : $\mathsf{REventClockTA} \to \mathsf{MEventClockTA}$ that given a recursive event-clock automaton A returns a monitored event-clock automaton B that accepts the same floating language. In the following, we apply T to a recursive event-clock automaton $A = (Q^A, Q_0^A, \delta_f^A, \delta_b^A, \mathcal{P}^A, \lambda^A, Q_{F_f}^A, Q_{F_b}^A)$, it returns a monitored event-clock automaton B.

- *Base case.* Let us first treat the basic case where A is a floating automaton. Then $B = (Q^B, Q_0^B, Q_M^B, \delta^B, \mathcal{P}^B, \lambda^B, Q_F^B)$ is a monitored floating automaton with the following elements:
  - *Locations.* The set of locations $Q^B = Q^A \times \{b, f, bf\}$, i.e. we take three copies of each locations of A and tag the first with b, the second with f and the third with bf. The locations tagged with b will be used to mimic the backwards runs, the locations tagged with f will be used to mimic the forward runs and, finally, the locations tagged with bf will be used to make the interface between forward and backward runs.
  - *Starting locations.* The set $Q_0^B = \{(q, b), (q, bf) | q \in Q_{F_b}^A\}$, of starting locations of the monitored automaton B are the final locations for the backward runs of the automaton A tagged with either b or bf.
  - *Monitored locations.* The set of monitored locations $Q_M = \{(q, bf) \mid q \in Q_0^A\}$, that is the set of monitored locations are locations that are the interface between backward and forward runs;
  - *Transition relation.* The transition relation δ^B of B is the union of the four following sets:
    1. $\{[(q_1, b), (q_2, b)] \mid (q_2, q_1) \in \delta_b^A\}$, i.e. two locations tagged with b are linked by the transition relation in B if they are linked by the backward transition relation in A; we reverse the direction of the transition as we are working with a forward transition relation in B;
    2. $\{[(q_1, f), (q_2, f)] \mid (q_1, q_2) \in \delta_f^A\}$, i.e. two locations tagged with f are linked by the transition relation in B if they are linked by the forward transition relation in A;
    3. $\{[(q_1, bf), (q_2, f)] \mid (q_1, q_2) \in \delta_f^A \text{ or } q_1 = q_2\}$, i.e. if the control of B is in a location tagged with bf, it can only evolve to locations tagged with f using the forward transition relation of A or evolve to the same location but tagged with f;
    4. $\{[(q_1, b), (q_2, bf)] \mid (q_2, q_1) \in \delta_b^A \text{ or } q_1 = q_2\}$, i.e. if the control of B is in a location tagged with b, it can only evolve to locations tagged with bf by using the inverse of the backward transition relation of A or it can evolve to the same location but tagged with bf;
  - *Propositions.* The set of propositions used by B is the same set of propositions used by A, i.e. $\mathcal{P}^B = \mathcal{P}^A$;

– *Labeling function.* The labeling of location $(q, -)$ in B is the same as the labeling of q in A, that is for all $(q, -) \in Q^B$, $\lambda^B((q, -)) = \lambda^A(q)$;

– *Acceptance condition.* The acceptance condition of B is defined by the following set of accepting locations: $\{(q, f) \mid q \in Q^A_{F_f}\}$, that is the same acceptance condition that the one for forward run in A.

Now let us prove that the floating language defined by the monitored floating automaton B is equal to the floating language defined by the floating automaton A.

– First, let us prove that if $(\kappa, t) \in \mathsf{FloatLang}(A)$ then $(\kappa, t) \in \mathsf{FloatLang}(B)$.

  If $(\kappa, t) \in \mathsf{FloatLang}(A)$ then we know that there exists an accepted backward run $\rho^b = (q^b_0, I^b_0)(q^b_1, I^b_1) \dots (q^b_n, I^b_n)$ and an accepted forward run $\rho^f = (q^f_0, I^f_0)(q^f_1, I^f_1) \dots (q^f_n, I^f_n) \dots$, furthermore we know that the backward run ends at time t while the forward run begins at time t and that $q^b_n = q^f_0$. Without loss of generality, see lemma 4.22, we can make the hypothesis that I^b_n and I^f_n are equal to $[t, t]$. Now, we define the run η as the concatenation of the three following sequences:

  * $\eta_1 = ((q^b_0, b), I_{b,0})((q^b_1, b), I^b_1) \dots ((q^b_{n-1}, b), I^b_{n-1})$. Intuitively, η_1 is the translation of the backward run in the b-tagged locations of B;
  * $\eta_2 = ((q^f_0, bf), [t, t])$. η_2 is just the location that makes the link between the part corresponding to the backward run at time t;
  * $\eta_3 = ((q^f_1, f), I^f_1)((q^f_2, f), I^f_1) \dots ((q^f_n, f), I^f_n) \dots$. Intuitively, η_3 is the translation of the forward run in the f-tagged locations of B.

  We now have to prove that the run $\eta = \eta_1 \cdot \eta_2 \cdot \eta_3$ is effectively an t-monitored accepted run for κ on B. For that, we check that η has the property of such a run:

  * *Monitoring.* By construction of η, we have $\eta(t) = (q^f_0, bf)$. As q^f_0 is the first location of the forward run ρ_f, we know that $q^f_0 \in Q^A_0$ which implies, by definition of Q^B_M that $(q^f_0, bf) \in Q^B_M$ and thus η is monitored at time t;
  * *Consecution.* We show that the consecution rule is verified for the 3 constituting part η_1, η_2, η_3 of η:
    · within η_1: let us consider the locations $(q^b_i, b), (q^b_{i+1}, b)$. By the consecution condition for A, we know that either $(q^b_{i+1}, q^b_i) \in \delta^A_b$ or $q^b_{i+1} = q^b_i$. The second case is trivial. In the first case, by construction of B, we obtain by point 1 of the definition of the transition relation of B that $[(q^b_i, b), (q^b_{i+1}, b)] \in \delta^B$ and thus the consecution condition is verified for η_1;
    · between η_1 and η_2: we must show that $[(q^b_{n-1}, b), (q^f_0, bf)] \in \delta^B$. We know that either $q^b_n = q^f_0$ (in the case of a stuttering step) or $(q^f_0, q^b_n) \in \delta^A_B$. In the two cases, we know that $[(q^b_{n-1}, b), (q^f_0, bf)] \in \delta^B$ by point 4 of the definition of δ^B.
    · We leave the two last cases, i.e. between η_2 and η_3 and within η_3, for the reader, there are treated in the same way as the two first cases.
  * *Constraints.* By the definition of λ^B and the construction of our run, it is easy to show that the constraints induced by η at each time t are exactly the same as the constraints induced by the backward and forward runs ρ_b and ρ_f. Thus the constraint condition is satisfied along κ as the constraint is satisfied for the backward and the forward runs.
  * *Accepting.* We know that the forward run $\rho_f = (q^f_0, I^f_0), (q^f_1, I^f_1), \dots$ respects the accepting condition imposed by A, that is there exists infinitely many positions $i \geq 0$ such that $q^f_i \in Q^A_F$. By construction η contains for each of those q^f_i a position (q^f_i, f), which belongs to Q^B_F by construction. And thus η is accepting.

– Second, let us prove that if $(\kappa, t) \in \mathsf{FloatLang}(B)$ then $(\kappa, t) \in \mathsf{FloatLang}(A)$. We know that there exists a t-monitored accepting run for κ on B. If we inspect the transition structure of automaton B, it is not difficult to see that the following property holds: a t monitored and accepted run must first traverse location tagged with b, reaches at time t a location tagged with bf and after this time t stays within locations tagged with f. We note such a run $\eta = ((q_0, b), I_0)((q_1, b), I_1)\dots((q_{n-1}, b), I_{n-1})((q_n, bf), I_n)((q_{n+1}, f), I_{n+1})\dots$, with $t \in I_n$. Without lose of generality, we can impose that $I_n = [t, t]$, since our automata are closed under stuttering refinement. Now, let us show how to construct a backward run ρ_b and a forward run ρ_f from this run η:

* we take $\rho^b = ((q_0, b), I_0)((q_1, b), I_1)\dots((q_n, b), I_n)$;
* and $\rho_f = ((q_n, bf), I_n)((q_{n+1}, f), I_{n+1})((q_{n+2}, f), I_{n+2})\dots$

It is routine to show that the constructed runs respect the conditions that allows us to conclude that $(\kappa, t) \in \mathsf{FloatLang}(A)$.

- *Inductive case.* By induction hypothesis, we know that for every $\mathsf{REventClockTA}$ C of level_j, with $0 \leq j < i$, we can construct a $\mathsf{MEventClockTA}$ D that accepts exactly the same floating language. In the sequel, we use the notation $T(C)$ to represent that congruent automaton. Let us show that we can construct an congruent $\mathsf{MEventClockTA}$ B for every $\mathsf{REventClockTA}$ A of level_i. The construction is similar to the one for the base case, except that we must handle properly real-time constraints and the labeling function. We detail those points:
  - *Atomic real-time constraints.* The set of atomic real-time constraints used in B is as follows: $\{z_{T(C)} \sim c \mid z_C \sim c \in \mathcal{A}^A\}$.
  - *Labeling function.* The labeling function of B is as for A except that each atomic real-time constraint $z_C \sim c$ is replaced by $z_{T(C)} \sim c$.

  The proof for the equivalence of floating languages is similar to the one for the base case.

□

We also have the reverse lemma:

Lemma 4.27 ($\mathsf{MEventClockTA} \subseteq \mathsf{REventClockTA}$) *For every monitored recursive event-clock automata A, we can construct a recursive event-clock automata B that accepts exactly the same floating language.*

Proof. This direction is simpler. We only treat the base case. The induction case is left to the reader. Let us consider a monitored floating automaton $A = (Q^A, Q_0^A, Q_M^A, \delta^A, \mathcal{P}^A, \lambda^A, Q_F^A)$, we construct a congruent floating automaton $B = (Q^B, Q_0^B, \delta_f^B, \delta_b^B, \mathcal{P}^B, \lambda^B, Q_{F_f}^B, Q_{F_b}^B)$ as follows:

- *Locations.* The set of locations Q^B is the same as in A, i.e. $Q^B = Q^A$;
- *Starting locations.* The set of starting location in B are the monitored locations of A, i.e. $Q_0 = Q_M$;
- *Forward and backward transition relations.* The forward transition relation of B is the transition relation of A, and the backward transition relation of B is the inverse of the transition relation of A, that is $\delta_f^B = \{(q_1, q_2) \mid (q_1, q_2) \in \delta^A\}$ and $\delta_b^A = \{(q_2, q_1) \mid (q_1, q_2) \in \delta^A\}$;
- *Propositions.* The set of propositions used by B is similar to the set of propositions used by A, i.e. $\mathcal{P}^B = \mathcal{P}^A$;
- *Labeling function.* The labeling function of B is as for A, that is for all locations $q \in Q^B$, $\lambda^B(q) = \lambda^A(q)$;
- *Forward and backward accepting locations.* The forward accepting locations of B are the accepting locations of A, that is $Q_{F_f}^B = Q_F^A$, and the backward accepting locations of B are the initial locations of A, i.e. $Q_{F_b}^B = Q_0^A$.

It is routine to prove that the constructed automaton B accepts the same floating language as A. □

This two last lemmas allow us to derive the theorem:

Theorem 4.28 (REventClockTA = MEventClockTA) *The class of recursive event-clock automata and monitored recursive event-clock automata are equally expressive.*

Now, we will concentrate on properties of monitored recursive event-clock automata. We will simply derive the appropriate corollaries for recursive event-clock automata.

4.4.1 Closure under Positive Boolean Operations

Let us now prove two first result about the closure property of monitored recursive event-clock automata: they are closed under positive boolean operations, i.e. closed under union and intersection.

Theorem 4.29 (MEventClockTA-**Union**) *Given two monitored recursive event-clock automata A and B defined on the same set of propositions, there always exists a third monitored recursive event-clock automaton C that accepts exactly the union of the timed floating languages of A and B, i.e.* $\mathsf{FloatLang}(C) = \mathsf{FloatLang}(A) \cup \mathsf{FloatLang}(B)$.

Proof. The proof is constructive. Let A and B be MEventClockTA, we construct the MEventClockTAC that accepts the union of the floating languages of A and B as follows:

- *Locations.* The set of locations of C are the tuples (q, Ξ) such that
  1. either $q \in Q^A$, $\Xi \in \mathsf{Limit}(\mathcal{P}^C \cup \mathcal{A}^C)$ and for all $\phi \in \mathsf{Limit}(\mathcal{P}^A \cup \mathcal{A}^A)$: $\phi \in \Xi$ iff $\phi \in \lambda^A(q)$, which will ensure the coherence of the labeling of (q, Ξ) with the labeling of q in A,
  2. or $q \in Q^B$, $\Xi \in \mathsf{Limit}(\mathcal{P}^C \cup \mathcal{A}^C)$ and for all $\phi \in \mathsf{Limit}(\mathcal{P}^B \cup \mathcal{A}^B)$: $\phi \in \Xi$ iff $\phi \in \lambda^B(q)$, which will ensure the coherence of the labeling of (q, Ξ) with the labeling of q in B.
- *Starting locations.* The subset of starting locations of C is the following set $Q_0^C = \{(q, \Xi) \in Q^C \mid q \in Q_0^A \text{ or } q \in Q_0^B\}$.
- *Monitored locations.* The subset of monitored locations of C is the following set: $Q_M^C = \{(q, \Xi) \in Q^C \mid q \in Q_M^A \text{ or } q \in Q_M^B\}$;
- *Transition relation.* The transition relation of C is the following subset of $Q^C \times Q^C$: $\delta^C = \{[(q_1, \Xi_1), (q_2, \Xi_2)] \mid (q_1, q_2) \in \delta^A \text{ or } (q_1, q_2) \in \delta^B\}$;
- *Propositions and atomic clock constraints.* The propositions in C are as in A, the atomic clock constraints used in C is the union of the atomic clock constraints used in A and B, that is $\mathcal{P}^C = \mathcal{P}^A = \mathcal{P}^B$,$\mathcal{A}^C = \mathcal{A}^A \cup \mathcal{A}^B$;
- *Labeling function.* The label of the location (q, Ξ) is simply the set of literals Ξ: $\lambda^C((q, \Xi)) = \Xi$, for every $(q, \Xi) \in Q^C$.
- *Accepting locations.* The accepting condition for C is the union of the accepting condition for A and B, that is $Q_F^C = \{(q, \Xi) \mid q \in Q_F^A \text{ or } q \in Q_F^B\}$;

It is direct to show that the constructed automaton accepts the desired floating language. □

By the equivalence between monitored and non monitored recursive event clock automata, see theorem 4.28, we have the following corollary:

Corollary 4.30 (REventClockTA-**union**) *Given two recursive event-clock automata A and B defined on the same set of propositions, there always exists a third recursive event-clock automaton C that accepts exactly the union of the floating real-time languages of A and B, i.e.* $\mathsf{FloatLang}(C) = \mathsf{FloatLang}(A) \cup \mathsf{FloatLang}(B)$.

We now turn to the closure of MEventClockTA to intersection. The following theorem states that MEventClockTA are closed under intersection:

Theorem 4.31 (MEventClockTA-**Intersection**) *Given two monitored recursive event-clock automata A and B defined on the same set of propositions, there always exists a third monitored recursive event-clock automaton C that accepts exactly the intersection of the floating real-time languages of A and B, i.e.* $\mathsf{FloatLang}(C) = \mathsf{FloatLang}(A) \cap \mathsf{FloatLang}(B)$.

Proof. Let A and B, we construct C that accepts the intersection of the timed floating languages of A and B as follows:

- *Locations.* The set of locations of C are the tuples (q^a, q^b) such that $q^a \in Q^A$, $q^b \in Q^B$ and for all literals $\phi \in \mathsf{Limit}(\mathcal{P}^A \cup \mathcal{A}^A) \cap \mathsf{Limit}(\mathcal{P}^B \cup \mathcal{A}^B)$, $\phi \in \lambda^A(q^a)$ iff $\phi \in \lambda^B(q^b)$. So the set of locations of C is the set of pairs of locations of A and B that have compatible labels.
- *Starting locations.* The set of starting locations of C is the following set $Q_0^C = \{(q^a, q^b) \in Q^C \mid q^a \in Q_0^A \text{ and } q^b \in Q_0^B\}$;
- *Monitored locations.* The set of monitored locations of C is the following subset of Q^C: $Q_M^C = \{(q^a, q^b) \in Q^C \mid q^a \in Q_M^A \text{ and } q^b \in Q_M^B\}$;
- *Transition relation.* The transition relation of C is the following subset of $Q^C \times Q^C$: $\delta^C = \{[(q_1^a, q_1^b), (q_2^a, q_2^b)] \mid (q_1^a, q_2^a) \in \delta^A \vee (q_1^a = q_2^a) \text{ and } (q_1^b, q_2^b) \in \delta^B \vee (q_1^b = q_2^b)\}$;
- *Propositions and Atomic real-time constraints.* The set of propositions used in C is the set of propositions used in A and B, the set of atomic real-time constraints is the union of the sets used in A and B, that is $\mathcal{P}^C = \mathcal{P}^A = \mathcal{P}^B$,$\mathcal{A}^C = \mathcal{A}^A \cup \mathcal{A}^B$;
- *Labeling function* The atom that labels a location (q^a, q^b) of C is the union (giving the conjunction of constraints) of the label of q^a in A and the label of q^b in B (remember that by definition (q^a, q^b) have compatible labels), that is $\lambda^C((q^a, q^b)) = \lambda^A(q^a) \cup \lambda^B(q^b)$, for every $(q^a, q^b) \in Q^C$;
- *Accepting locations.* For the accepting condition, we define a generalized Büchi condition: $Q_F^C = \{F_A, F_B\}$, with $F_A = \{(q^a, q^b) \mid q^a \in Q_F^A\}$ and $F_B = \{(q^a, q^b) \mid q^b \in Q_F^B\}$. This generalized Büchi acceptance condition can be converted into a Büchi acceptance condition using the usual technique. This costs only a doubling of the number of locations.

It is direct to show that the constructed automaton accepts the desired floating language. □

Again, by theorem 4.28, we obtain the following corollary:

Corollary 4.32 (REventClockTA-**Intersection**) *Given two recursive event-clock automaton A and B defined on the same set of propositions, there always exists a third recursive event-clock automaton C that accepts exactly the intersection of the floating timed languages of A and B, i.e.* $\mathsf{FloatLang}(C) = \mathsf{FloatLang}(A) \cap \mathsf{FloatLang}(B)$.

4.4.2 Closure under Negation

Let us now turn to the problem of complementing monitored recursive event-clock automata. The problem is more complicated. By inspecting the definition of run for our MEventClockTA, we can see that the problem of floating acceptance can be decomposed into two usual forward acceptances. In fact, a MEventClockTA accepts (κ, t) if it has a finite run on the prefix $(\kappa, [0, t])$ that ends in a monitored location q_m and a run on the suffix $(\kappa, [t..\infty))$ that is accepting and starts in q_m.

To formalize this intuition, we define two new types of languages for MEventClockTA. First, *prefix acceptance* allows us to define the *prefix real-time language* of a MEventClockTA A, noted $\mathsf{PreLang}(A)$. This language, again, is a set of pairs (κ, t) where κ is a TSS and $t \in \mathbb{R}^+$. The intuition behind this language is that if $(\kappa, t) \in \mathsf{PreLang}(A)$ then there exists a finite run ρ of length t of A such that the run begins at time 0 in a starting location, ends at time t in a monitored location and the constraints that are induced by the run are verified by the TSS κ. Second, *suffix acceptance* allows us to define the *suffix real-time language* of a MEventClockTA, noted $\mathsf{SufLang}(A)$. This language is also a set of pairs (κ, t) where κ is a TSS and $t \in \mathbb{R}^+$. Here, the intuition is that if $(\kappa, t) \in \mathsf{SufLang}(A)$ then there exists a infinite run ρ of A on κ, such that the run begins at time t in a monitored location, goes through accepting locations infinitely often and the constraints that are induced by the run are verified by the TSS κ. The PreLang and SufLang will be assembled in lemma 4.46. Let us now define formally PreLang and SufLang:

Definition 4.33 (Prefix Language) Given a monitored recursive event-clock automaton $A = (Q, Q_0, Q_M, \delta, \mathcal{P}, \mathcal{A}, \lambda, Q_F)$, a pair (κ, t) belongs to the $\mathsf{PreLang}(A)$ iff there exists a finite timed run $\rho = (q_0, I_0), (q_1, I_1), \ldots, (q_n, I_n)$ such that:

Covering The run ρ covers time up to t, i.e. $\bigcup_{i=0}^{i=n} I_i = [0, t]$;

Starting The run ρ starts in a starting location of A, that is $q_0 \in Q_0$;

Consecution The run ρ respects the transition relation of A, i.e. for all positions i such that $1 \leq i < n$, we have that $(q_i, q_{i+1}) \in \delta$ or $q_i = q_{i+1}$ (stuttering steps are allowed);

Constraints The TSS κ respects the constraints induced by ρ, that is for all time $t' \in [0, t]$: $(\kappa, t') \models \lambda(\rho(t'))$;

Further, the run ρ is *accepting* if it ends in a monitored location of A, i.e. $q_n \in Q_M$. □

Definition 4.34 (Suffix Language) Given a monitored recursive event-clock automaton $A = (Q, Q_0, Q_M, \delta, \mathcal{P}, \mathcal{A}, \lambda, Q_F)$, a pair (κ, t) belongs to $\mathsf{SufLang}(A)$ iff there exists an infinite timed run $\rho = (q_0, I_0), (q_1, I_1), \ldots, (q_n, I_n), \ldots$ such that:

Covering The run ρ covers time from t, i.e. $\bigcup_{i=0}^{i=\omega} I_i = [t, \infty)$;

Starting The run ρ starts in a monitored location of A, that is $q_0 \in Q_M$;

Consecution The run ρ respects the transition relation of A, i.e. for all positions $i \geq 0$ we have that $(q_i, q_{i+1}) \in \delta$ or $q_i = q_{i+1}$ (stuttering steps are allowed);

Constraints The TSS κ respects the constraints induced by ρ, that is for all time $t' \in [t, \infty)$: $(Tss, t') \models \lambda(\rho(t'))$.

The run ρ is accepting if it intersects infinitely often with the set of accepting locations, i.e. there exists infinitely many positions i such that $q_i \in Q_F$. □

Next we show that MEventClockTA are determinizable and keep, in their deterministic version, their expressive power for defining prefix languages. First, let us define formally the notion of deterministic and total monitored recursive event-clock automata.

Definition 4.35 (Deterministic and Total MEventClockTA) A monitored recursive event-clock automaton $A = (Q, Q_0, Q_M, \delta, \mathcal{P}, \mathcal{A}, \lambda, Q_F)$ is *deterministic* iff the following conditions are satisfied:

Unique initial locations All pairs of initial locations have different (and thus mutually non satisfiable labels), that is, for all $q_1, q_2 \in Q_0$, with $q_1 \neq q_2$, $\lambda(q_1) \neq \lambda(q_2)$.

Unique next location Given a location q_1, all successor locations of q_1 have different labels, i.e. for all q_2, q_3 such that $(q_1, q_2) \in \delta$ and $(q_1, q_3) \in \delta$ then if $q_2 \neq q_3$ then $\lambda(q_2) \neq \lambda(q_3)$. As labels are set of literals that are true when the control resides in the location, as all successor locations of a location q_1 have different labels and thus mutually non satisfiable labels, the possible successor location in a run is unique;

Non repeating For every location q, the labels of its next locations are all different from the one of q, i.e. for every $q \in Q$, for every q' such that $q' \neq q$ and $(q, q') \in \delta$, $\lambda(q) \neq \lambda(q')$.

Furthermore, we say that A is *total* iff the following condition is satisfied:

Totality The two following points must be verified:

1. For every $\Xi \in 2^{\mathsf{Limit}(\mathcal{P} \cup \mathcal{A})}$, there exists an initial location q whose label is Ξ, that is $q \in Q_0$ and $\lambda(q) = \Xi$;
2. For every location $q_1 \in Q$, for every $\Xi \in 2^{\mathsf{Limit}(\mathcal{P} \cup \mathcal{A})}$ there exists a location q_2 such that either $q_1 = q_2$ or $(q_1, q_2) \in \delta$, and $\lambda(q_2) = \Xi$.

□

"Unique initial location" and "unique next location" conditions ensure that there exists at most one, up to stuttering, prefix run (maybe non accepting) for every pair (κ, t) on a deterministic monitored event-clock automaton. The condition "non repeating" imposes that two consecutive locations in a deterministic automaton can not be labeled with the same literals. This is important and necessary because we are considering automata that evolves along (continuous) timed state sequences and if two consecutive locations are labeled with the same (open) label, the automaton can change from one location to the next nondeterministically at any time of an open interval that agrees with the label, making the automaton non deterministic. "Totality" imposes that every pair (κ, t) has one prefix (not necessarily accepting) run on the monitored event-clock automaton.

The usual subset construction does not work when directly applied to $\mathsf{MEventClockTA}$. If the usual subset construction is applied without care, the automaton obtained could contain two consecutive locations with the same label and, thus, would violate the "non repeating condition" and thus not be deterministic. Before applying the subset construction, we apply to the automaton a transformation that is exposed in the following lemma and its proof.

Lemma 4.36 (Non Repeating $\mathsf{MEventClockTA}$) *For every monitored recursive event-clock automata A, there exists an equivalent monitored event-clock automata B that accepts the same anchored, floating, prefix and suffix languages and that have the property that it does not have any two consecutive locations labeled identically, that is, there does not exists $q_1, q_2 \in Q^B$ with $q_1 \neq q_2$ such that $(q_1, q_2) \in \delta^B$ and $\lambda^B(q_1) = \lambda^B(q_2)$.*

Proof. First note that if two locations q_1, q_2 are labeled by singular sets of literals (see definition 2.19), and linked by an edge, i.e. $(q_1, q_2) \in \delta^A$, then we can suppress this edge without changing the languages (anchored and floating) defined by the automaton A. In fact, this edge can not be used by any run. As q_1, q_2 are labeled with a singular literals, the control can only stay there during a singular interval of time. But two singular interval of time can not follow each other in a sequence of intervals. We can also suppress edges between two locations that are labeled by two different open sets of literals. Suppose that we have a portion of a TSS where an open label is true. From the definition of open label, it is direct to prove that this portion of the TSS must be an open interval of time. So let us consider that the open label λ is true during the open interval of time (a, b). If the control in time $t \in (a, b)$ is in a location with label λ then the control can take any amount of transitions to reach other locations labeled with λ before leaving the interval (a, b). This intuition is formalized by the following functions:

- $\mathsf{SReach}_A : Q^A \to 2^{Q^A}$, this function, when applied to a location q returns all the locations that can be reached from q in the transition structure of A only by using locations labeled as q. Formally, the function is defined as follows: $q' \in \mathsf{SReach}_A(q)$ iff there exists a sequence of locations $q_0, q_1, \ldots, q_n$ such that

   1. $n \geq 0$;
   2. $q_0 = q$;
   3. $q_n = q'$;
   4. for all positions i, $0 \leq i < n$, either $q_i = q_{i+1}$ or $(q_i, q_{i+1}) \in \delta^A$, and $\lambda^A(q_i) = \lambda^A(q)$;

- $\mathsf{SReachMoni}_A : Q^A \to 2^{Q^A}$, this function, when applied to a location q returns all the locations that can be reached from q in the transition structure of A by using only locations labeled as q and by passing at least by a monitored location. Formally, the function is defined as follows: $q' \in \mathsf{SReachMoni}_A(q)$ iff there exists a sequence of locations $q_0, q_1, \ldots, q_n$ such that

   1. $n \geq 0$;
   2. $q_0 = q$;
   3. $q_n = q'$;
   4. for all positions i, $0 \leq i < n$, either $q_i = q_{i+1}$ or $(q_i, q_{i+1}) \in \delta^A$, and $\lambda^A(q_i) = \lambda^A(q)$; and
   5. there exists a position i, $0 \leq i < n$ such that $q_i \in Q^A_M$;

- $\mathsf{SReachAcc}_A : Q^A \to 2^{Q^A}$, this function, when applied to a location q returns all the locations that can be reached from q in the transition structure of A by using only locations labeled as q and by passing at least by an accepting location. Formally, the function is defined as follows: $q' \in \mathsf{SReachAcc}_A(q)$ iff there exists a sequence of locations $q_0, q_1, \ldots, q_n$ such that

   1. $n \geq 0$;
   2. $q_0 = q$;
   3. $q_n = q'$;
   4. for all positions i, $0 \leq i < n$, either $q_i = q_{i+1}$ or $(q_i, q_{i+1}) \in \delta^A$, and $\lambda^A(q_i) = \lambda^A(q)$; and
   5. there exists a position i, $0 \leq i < n$ such that $q_i \in Q^A_F$;

We construct the $\mathsf{MEventClockTA}$ $B = (Q^B, Q^B_0, Q^B_M, \delta^B, \mathcal{P}^B, \mathcal{A}^B, \lambda^B, Q^B_F)$ as follows:

- *Locations.* The locations of B will the set of 3-tuples (q, η, ξ) such that:
   - $q \in Q^A$;
   - $\eta \in \{M, \overline{M}\}$ and if $\kappa = M$ then $\mathsf{SReachMoni}_A(q) \neq \emptyset$, that is q can access a monitored location by staying on locations that are labeled with the same open label;
   - $\xi \in \{F, \overline{F}\}$ and if $\xi = F$ then $\mathsf{SReachAcc}_A(q) \neq \emptyset$, that is q can access an accepting location by staying on locations that are labeled with the same open label.
- *Initial locations.* The set of initial locations Q^B_0 is the set of locations $(q, \eta, \xi) \in Q^B$ with $q \in Q^A_0$, that is the tuples whose location q is an initial location of A;
- *Monitored locations.* The set of monitored locations Q^B_M is the set of locations $(q, \eta, \xi) \in Q^B$ with $\eta = M$, that is the tuples whose locations q can access, by staying on locations with the same label as q, a monitored location.
- *Transition relation.* A pair $[(q_1, \eta_1, \xi_1), (q_2, \eta_2, \xi_2)]$ belongs to the transition relation δ^B iff the four following rules are verified:

   1. if $\eta_1 = \overline{M}$ and $\xi_1 = \overline{F}$ then $q_2 \in \mathsf{SReach}_A(q_1)$ and $\lambda^A(q_2) \neq \lambda^A(q_1)$;

2. if $\eta_1 = M$ and $\xi_1 = \overline{F}$ then $q_2 \in \mathsf{SReachMoni}_A(q_1)$ and $\lambda^A(q_2) \neq \lambda^A(q_1)$;
3. if $\eta_1 = \overline{M}$ and $\xi_1 = F$ then $q_2 \in \mathsf{SReachAcc}_A(q_1)$ and $\lambda^A(q_2) \neq \lambda^A(q_1)$;
4. if $\eta_1 = M$ and $\xi_1 = F$ then $q_2 \in \mathsf{SReachMoni}_A(q_1) \cap \mathsf{SReachAcc}_A(q_1)$ and $\lambda^A(q_2) \neq \lambda^A(q_1)$;

- *Propositions and atomic real-time constraints.* The set of propositions and of atomic real-time constraints used in B is the same as the ones used in A, that is $\mathcal{P}^B = \mathcal{P}^A$ and $\mathcal{A}^A = \mathcal{A}^B$;
- *Labeling function.* The labeling function of B is derived from the labeling function of A as follows: for all $(q, \eta, \xi) \in Q^B$, $\lambda^B((q, \eta, \xi)) = \lambda^A(q)$;
- *Accepting locations.* The set of accepting locations Q^B_F is the subset of locations $(q, \eta, \xi) \in Q^B$ such that $\xi = F$.

It is routine to establish that the floating language of B is exactly the same as the floating language of A. □

The next theorem states that every monitored event-clock automaton with the non-repeating property, can be determinised.

Lemma 4.37 ($\mathsf{MEventClockTA}$**-Determinization**) *For every monitored event-clock automaton A with the non repeating property, one can construct a deterministic and total monitored event-clock automaton C that accepts the same prefix language, i.e.* $\mathsf{PreLang}(A) = \mathsf{PreLang}(C)$.

Proof. Our proof is constructive. Let us consider A and construct the deterministic forward event-clock automaton B as follows:

- *Locations.* The set of locations of B is the set of non-empty subsets of locations of A that share the same label, that is: $\{q_1, \ldots, q_n\} \in Q^B$ iff
  1. for all i, $1 \leq i \leq n$: $q_i \in Q^A$ (subset of Q^A).
  2. $n \geq 1$ (non-empty subset);
  3. for all i, j such that $1 \leq i < j \leq n$, we have that $\lambda^A(q_i) = \lambda^A(q_j)$ (same label).
- *Propositions and atomic real-time constraints.* The set of propositions used in B is the same as the set of propositions used in A, i.e. $\mathcal{P}^B = \mathcal{P}^A$, the set of atomic real-time constraints used in B is the same as the set of real-time constraints used in A, i.e. $\mathcal{A}^B = \mathcal{A}^A$;
- *Labeling function.* The labeling function is defined as follows: $\lambda^B(l) = \lambda^A(q)$ with $q \in l$, for all $l \in Q^B$. Recall that the locations appearing in l are all labeled with the same label in A, we just take this label for l.
- *Starting locations.* The set of starting locations of B is the subset of locations that contains only initial locations of A, expressed by point 1 below, and that are maximal for their label, expressed by point 2, that is $l \in Q^B_0$ iff
  1. for all $q \in l$, $q \in Q^A_0$, and
  2. there does not exists a location l' with
     (a) $\lambda^B(l') = \lambda^B(l)$,
     (b) for all $q \in l'$, $q \in Q^A_0$ and
     (c) $l \subset l'$;
- *Monitored locations.* A location $l \in Q^B$ belongs to the set Q^B_M of monitored locations iff there exists a location of A in l that is monitored, i.e. $l \in Q^B_M$ iff there exists $q \in l$ such that $q \in Q^A_M$.
- *Transition relation.* We have that $(l_1, l_2) \in \delta^B \subseteq Q^B \times Q^B$ iff

1. for all $q_2 \in l_2$, there exists $q_1 \in l_1$ such that $(q_1, q_2) \in \delta^A$;
2. for all $q_2 \in Q^A$ such that $\lambda^A(q_2) = \lambda^B(l_2)$ and such that there exists $q_1 \in l_1$ with $(q_1, q_2) \in \delta^A$, we have $q_2 \in l_2$;

In words, the point (1) says that locations in l_2 are δ^A-successors of locations in l_1 and (2) says that l_2 is the maximal set of locations that share the label of l_2 and are δ^A-successors of a location of l_1.

- *Accepting locations.* As we are only interested in the prefix language of B, we take arbitrarily $Q^B_F = Q^B$.

It is not difficult to show that $(\kappa, t) \in \mathsf{PreLang}(B)$ iff $(\kappa, t) \in \mathsf{PreLang}(A)$. Now, let us see how we can transform B into a deterministic automaton C that has the totality property. We construct C as follows:

- *Locations.* We take $Q^C = Q^B \cup D$, where D is a set of dummy locations. We take one dummy locations for each possible label in B, that is $D = \{l \mid l \in 2^{\mathsf{Limit}(\mathcal{P}^B \cup \mathcal{A}^B)}\}$. The locations of D, will be used to handle the pairs (κ, t) that does not belong to the prefix language of B.
- *Starting locations.* We take $Q^C_0 = Q^B_0 \cup D_{init}$, where $D_{init} = \{q \in D | \nexists q' \in Q^B_0 : \lambda^B(q') = q\}$. So D' contains locations that correspond to labels for which there does not exists an initial location in B;
- *Monitored locations.* $Q^C_M = Q^B_M$, the monitored locations are the monitored locations of B, no dummy locations is monitored.
- *Transition relation.* The transition relation $\delta^C \subseteq Q^C \times Q^C$ is the set of pairs that respects the following conditions:
  1. for all $q_1, q_2 \in Q^B$: $(q_1, q_2) \in \delta^C$ iff $(q_1, q_2) \in \delta^B$;
  2. for all $q_1, q_2 \in D$ with $q_1 \neq q_2$: $(q_1, q_2) \in \delta^C$;
  3. for all $q_1 \in Q^B$, $q_2 \in D$: $(q_1, q_2) \in \delta^C$ iff $\lambda^B(q_1) \neq q_2$ and there does not exist $q_3 \in Q^B$ such that $\lambda(q_3) = q_2$ and $(q_1, q_3) \in \delta^B$;
  4. for all $q_1 \in D$, $q_2 \in Q^B$: $(q_1, q_2) \notin \delta^C$.

  Condition 1 ensures that the transitions possible in B are possible in C and vice versa; Condition 2 guarantees that when in a location of D the transition does no more constraint the possible runs; Condition 3 says that we can go from a location of B to a dummy location if and only if the transition is not possible in B for a given label; Condition 4 ensures that when in a dummy location it is not possible to return into the locations of B.
- *Propositions and atomic constraints.* $\mathcal{P}^C = \mathcal{P}^B$ and $\mathcal{A}^C = \mathcal{A}^B$, the propositions and atomic clock constraints used in C are similar to the ones used in B;
- *Labeling function.* λ^C is defined as follows:
  - for $q \in Q^B$, $\lambda^C(q) = \lambda^B(q)$;
  - for $q \in D$, $\lambda^C(q) = q$.

  Thus the labels of locations of B are conserved and the labels of dummy locations are simply the set of literals that constitutes the locations.
- *Accepting locations.* As for B, we take arbitrarily $Q^C_F = Q^C$.

Again, it is easy to show that the prefix language of B is preserved by C. □

Corollary 4.38 *Let A, B and C as in the last lemma For every* TSS $\kappa \in \mathsf{TSS}(\mathcal{P}^A)$, *there exists one run ρ (up to stuttering)* [2] *of C on κ, and the following property is verified: if $\rho(t) \in Q^B$ then there exists, for each $q \in \rho(t)$, a prefix run ρ^q on κ in A that covers $[0, t]$ and ends-up in location q, that is $\rho^q(t) = q$.* □

We will use this last corollary in the construct for the emptiness of monitored recursive event-clock automata.

[2]Note that here we identify two runs if they only differ by stuttering steps, we are only interested in the function $\rho : \mathbb{R}^+ \to Q^C$.

Complement of the Prefix Language We are now able to prove that monitored event-clock automata are closed under negation for their prefix languages.

Lemma 4.39 (Prefix Complement) *For every monitored recursive event-clock automaton A, we can construct another monitored recursive event-clock automaton B that accepts exactly the complement of the prefix language of A, i.e.* $\mathsf{PreLang}(B) = \overline{\mathsf{PreLang}(A)}$.

Proof. As noted in corollary 4.38, a pair (κ, t) has always one and only one prefix run on C, the deterministic and total version of A. In that case only the monitoring condition determines if a pair (κ, t) belongs to the prefix language of C. Thus to complement the prefix language of A, we just have to complement the monitoring condition of C. So we construct B as follows. First compute C as in lemma 4.37. Second, we take B as C except for the monitored locations where we take $Q^B_M = Q^C \setminus Q^C_M$. The constructed automaton B accepts $\overline{\mathsf{PreLang}(A)}$, the desired language. □

Complement of the Suffix Language Complement the suffix language of a $\mathsf{MEventClockTA}$ is more difficult. The difficulty has nothing to do with the fact that we are working with real-time automata because we are considering a Büchi acceptance condition. For such acceptance condition, it is well known that the usual subset construction does not work []. Instead of "re-doing" all the proofs for the complementation of Büchi automata, we show how to reduce our problem of complementation to the problem of complementation of usual Büchi automata on ω-sequences.

To relate a pair (κ, t) to a ω-sequence $\overline{\gamma} = \gamma_1\gamma_2 \ldots \gamma_n \ldots$, we use a function, called α and defined as follows:

Definition 4.40 (Function α) Given a TSS κ, a time $t \in \mathbb{R}^+$, the set of propositions $\mathcal{P}$ on which $\kappa = (\overline{\sigma}, \overline{I})$ is defined and a set of atomic clock constraints $\mathcal{A}$, $\alpha(\kappa, t, \mathcal{P}, \mathcal{A})$ returns the infinite sequence $\overline{\gamma}$ defined on the set of proposition $\mathcal{P}' = \{p_\phi \mid \phi \in \mathsf{Limit}(\mathcal{P} \cup \mathcal{A})\}$ such that: if $\kappa' = (\overline{\sigma}', \overline{I}')$ is the coarsest $\mathsf{Limit}(\mathcal{P} \cup \mathcal{A}) - \mathsf{Fine}$ TSS that refines κ, and $t \in I'_i$:

$$\gamma_j = \{p_\phi \in \mathcal{P}' \mid (\kappa', t') \models \phi, \text{ for all } t' \in I'_{i+j}\}$$

That is, γ_j contains all propositions associated with literals of $\mathsf{Limit}(\mathcal{P} \cup \mathcal{A})$ that are true during the interval I'_{i+j} of κ'. □

Note that for every TSS κ, there exists only one coarsest $\mathsf{Limit}(\mathcal{P}^A \cup \mathcal{A}^A) - \mathsf{Fine}$ TSS and thus $\overline{\gamma}$ is unique for every pair (κ, t).

The idea of the reduction is depicted in figure 4.4.2 and is decomposed in three steps:

(1) Given an $\mathsf{MEventClockTA}$ A, defined on the set of propositions $\mathcal{P}$ and atomic clock constraints $\mathcal{A}$, we construct a Büchi automaton B that accepts a language that respects the following property:

$$\text{for all } \kappa \in \mathsf{TSS}(\mathcal{P}), \text{ for all time } t \in \mathbb{R}^+,$$
$$(\kappa, t) \in \mathsf{SufLang}(A) \text{ iff } \alpha(\kappa, t, \mathcal{P}, \mathcal{A}) \in \mathsf{AncLang}(B)$$

(2) As the formalism of Büchi automata is closed under negation [SVW85], see theorem **??**, we can construct C such that:

$$\text{for all } \kappa \in \mathsf{TSS}(\mathcal{P}), \text{ for all time } t \in \mathbb{R}^+,$$
$$(\kappa, t) \in \mathsf{SufLang}(A) \text{ iff } \alpha(\kappa, t, \mathcal{P}, \mathcal{A}) \in \mathsf{AncLang}(B) \text{ iff } \alpha(\kappa, t, \mathcal{P}, \mathcal{A}) \notin \mathsf{AncLang}(C)$$

(3) From C, it remains to construct a $\mathsf{MEventClockTA}$ D such that:

$$\text{for all } \kappa \in \mathsf{TSS}(\mathcal{P}), \text{ for all time } t \in \mathbb{R}^+,$$
$$(\kappa, t) \in \mathsf{SufLang}(D) \text{ iff } \alpha(\kappa, t, \mathcal{P}, \mathcal{A}) \in \mathsf{AncLang}(C)$$

$$\begin{tikzcd} A \arrow[r, "(1)"] & B \arrow[d, "(2)"] \\ D & C \arrow[l, "(3)"] \end{tikzcd}$$

Figure 3: Complement of the suffix language

This automaton D accepts exactly the desired language, that is, $\mathsf{SufLang}(D) = \overline{\mathsf{SufLang}(A)}$.

The following two lemmas expressed that the transformation (1) and (3) are indeed possible:

Lemma 4.41 (From MEventClockTA to BA) *Given an* MEventClockTA A *that uses the set of propositions* $\mathcal{P}^A$ *and atomic clock constraints* $\mathcal{A}^A$*, we can construct a Büchi automaton* B *on the set of propositions* $\mathcal{P}^B = \{p_\phi \mid \phi \in \mathsf{Limit}(\mathcal{P}^A \cup \mathcal{A}^A)\}$ *such that:*

$$\textit{for all } \kappa \in \mathsf{TSS}(\mathcal{P}^A), \textit{ for all time } t \in \mathbb{R}^+, \\ (\kappa, t) \in \mathsf{SufLang}(A) \textit{ iff } \alpha(\kappa, t, \mathcal{P}^A, \mathcal{A}^A) \in \mathsf{AncLang}(B)$$

Proof.(sketch) By lemma 4.36, we can make the hypothesis that A has the non-repeating property. In that case, the Büchi automaton B can simply be obtain from A by:

- taking the same set of locations;
- adding to the transition relation all pairs (q, q), for all locations q, this is because, the notion of run in MEventClockTA allows stuttering steps;
- the initial locations of B are the monitored locations of A;
- adapting the labels are as follows: $p_\phi \in \lambda^B(q)$ iff $\phi \in \lambda^A(q)$, for all $\phi \in \mathsf{Limit}(\mathcal{P}^A \cup \mathcal{A}^A)$ and for all locations q;

It is not difficult to prove that the constructed Büchi automaton B accepts precisely the desired language. □

Conversely, we have

Lemma 4.42 (From BA to MEventClockTA) *Given a Büchi automaton* C *on the set of propositions* $\mathcal{P}^C = \{p_\phi \mid \phi \in \mathsf{Limit}(\mathcal{P}^D \cup \mathcal{A}^D)\}$*, we can construct an* MEventClockTA D *that uses the set of propositions* $\mathcal{P}^D$ *and atomic clock constraints* $\mathcal{A}^D$*,such that:*

$$\textit{for all } \kappa \in \mathsf{TSS}(\mathcal{P}^D), \textit{ for all time } t \in \mathbb{R}^+, \\ (\kappa, t) \in \mathsf{SufLang}(D) \textit{ iff } \alpha(\kappa, t, \mathcal{P}^D, \mathcal{A}^D) \in \mathsf{AncLang}(C)$$

Proof.(sketch) Again, the transformation is very simple. The MEventClockTA D is constructed from the BA C by:

- taking the same set of locations and the same transition relation;
- taking the initial locations of C as the monitored locations of D;
- adapting the labels as follows: $\phi \in \lambda^D(q)$ iff $p_\phi \in \lambda^C(q)$, for all $\phi \in \mathsf{Limit}(\mathcal{P}^D \cup \mathcal{A}^D)$, for all locations q;

It is not difficult to prove that the constructed automaton accepts the right suffix language. □

The construction that we have presented above allows us to derive the following lemma:

Lemma 4.43 (Suffix Complement) *For every monitored recursive event-clock automaton A, we can construct another monitored recursive event-clock automaton B that accepts exactly the complement of the suffix language of A, i.e.* $\mathsf{SufLang}(B) = \overline{\mathsf{SufLang}(A)}$. □

It is important to note that the proposed construction only works because to every tuple $(\kappa, t, \mathcal{P}, \mathcal{A})$ corresponds exactly one ω-sequence $\overline{\gamma}$. This is because the value of each event-clock is determined by κ at all time $t \in \mathbb{R}^+$ and consequently, the truth value of the atomic clock constraints of $\mathcal{A}$ is also determined by κ at all time $t \in \mathbb{R}^+$ and not by a particular run of the automaton on κ. Thus the proposed construction does not work for timed automaton (and it is not a surprise, for timed automata are not closed under negation). In a timed automaton, the value of a clock along a TSS κ does not only depend on the TSS κ but also on the particular run that is chosen to read κ. So to each tuple $(\kappa, t, \mathcal{P}, \mathcal{A})$ corresponds a set of ω-sequence $\overline{\gamma}$, one for each possible run.

Complement of the Floating Language So far, we have shown how we can complement the prefix et suffix languages accepted by an MEventClockTA. Let us now turn to the problem of complementing the floating language accepted by a MEventClockTA. First, let us consider the following lemma:

Lemma 4.44 (Decomposition Monitored Condition) *The floating language accepted by a* MEventClockTA $A = (Q^A, Q_0^A, Q_M^A, \delta^A, \mathcal{P}^A, \mathcal{A}^A, \lambda^A, Q_F^A)$ *with* $Q_M^A = \{q_1, q_2, \ldots, q_m\}$ *can be expressed by the union of the floating languages of a collection* $A_1, A_2, \ldots, A_m$ *of* m MEventClockTA *that have an unique monitored location.*

Proof. We take each A_i identical to A except for the monitored locations: for $Q_M^{A_i}$, we take the singleton $\{q_i\}$. If $(\kappa, t) \in \mathsf{FloatLang}(A)$ then A has a q_j-t-monitored and accepted run ρ on (κ, t), for some j, $1 \leq j \leq m$. By construction of each A_i, ρ is also a monitored and accepted run of A_j on (κ, t) implying that $(\kappa, t) \in \mathsf{FloatLang}(A_j)$ and thus $(\kappa, t) \in \bigcup_{i=1}^{i=m} \mathsf{FloatLang}(A_i)$. Conversely if A_j has a monitored and accepted run ρ on (κ, t) then ρ is an q_j-t-monitored and accepted run of A on (κ, t) and thus $(\kappa, t) \in \mathsf{FloatLang}(A)$. □

Note that if A has only one monitored location, we have the following interesting property:

Lemma 4.45 (Unique Monitored Location) *Let A be an monitored recursive event-clock automaton with only one monitored location, that is $|Q_M| = 1$ then* $\mathsf{FloatLang}(A) = \mathsf{PreLang}(A) \cap \mathsf{SufLang}(A)$.

Proof. Let us assume that $Q_M = \{q_m\}$. We first prove that if $(\kappa, t) \in (\mathsf{PreLang}(A) \cap \mathsf{SufLang}(A))$ then $(\kappa, t) \in \mathsf{FloatLang}(A)$. As $(\kappa, t) \in \mathsf{PreLang}(A)$, we know that there exists a finite prefix run ρ^p of A on κ that ends at time t in location q_m, the unique monitored location of A. Similarly, as $(\kappa, t) \in \mathsf{SufLang}(A)$, we know that there exists an infinite suffix run ρ^s of A on κ that starts, at time t, in location q_m, the unique monitored location of A. The concatenation of ρ^p and ρ^s is a t-monitored and accepted run of A on (κ, t) and thus $(\kappa, t) \in \mathsf{FloatLang}(A)$. We now turn to the other direction. If $(\kappa, t) \in \mathsf{FloatLang}(A)$ then we know that there exists a t-monitored and accepted run on κ and thus $\rho(t) = q_m$. We simply decompose ρ into $\rho^{[0,t]}$ and $\rho^{[t,\infty]}$, where $\rho^{[0,t]}$ is the prefix of ρ up to time t and $\rho^{[t,\infty]}$ is the suffix of ρ that starts at time t. It is easy to show that $\rho^{[0,t]}$ is an accepted prefix run of A on (κ, t) and thus $(\kappa, t) \in \mathsf{PreLang}(A)$ and $\rho^{[t,\infty]}$ is a suffix run of A on (κ, t) and thus $(\kappa, t) \in \mathsf{SufLang}(A)$. □

Thus for an $\mathsf{MEventClockTA}$ with only one location, the problem is nearly solved. In fact, last lemma tells us that if A has only one monitored location, $\mathsf{FloatLang}(A) = \mathsf{PreLang}(A) \cap \mathsf{SufLang}(A)$ and thus $\overline{\mathsf{FloatLang}(A)} = \overline{\mathsf{PreLang}(A)} \cup \overline{\mathsf{SufLang}(A)}$. We already know how to obtain $\overline{\mathsf{PreLang}(A)}$ and $\overline{\mathsf{SufLang}(A)}$. It just remains us to show how given an $\mathsf{MEventClockTA}$ that accepts $\overline{\mathsf{PreLang}(A)}$ how to construct a automaton B such that $\mathsf{FloatLang}(B) = \overline{\mathsf{PreLang}(A)}$ and similarly for the automaton accepting $\overline{\mathsf{SufLang}(A)}$.

Lemma 4.46 (Complement Unique Monitored Location) *For every monitored event-clock automaton $A_m = (Q^{A_m}, Q_0^{A_m}, Q_M^{A_m}, \delta^{A_m}, \mathcal{P}^{A_m}, \mathcal{A}^{A_m}, \lambda^{A_m}, Q_F^{A_m})$ with a single monitored location q_m, we can compute a monitored event-clock automaton B that accepts the complement of the floating language of A_m.*

Proof. From lemma 4.45, we know that $\overline{\mathsf{FloatLang}(A)} = \overline{\mathsf{PreLang}(A)} \cap \overline{\mathsf{SufLang}(A)}$ and by lemma 4.39, we can construct a $\mathsf{MEventClockTA}$ B such that $\mathsf{PreLang}(B) = \overline{\mathsf{PreLang}(A)}$ and by lemma 4.43 a $\mathsf{MEventClockTA}$ C such that $\mathsf{SufLang}(C) = \overline{\mathsf{SufLang}(A)}$. As $\mathsf{MEventClockTA}$ are closed under intersection, see lemma 4.31, it remains to construct from B a $\mathsf{MEventClockTA}$ E such that $\mathsf{FloatLang}(E) = \mathsf{PreLang}(B)$ and a $\mathsf{MEventClockTA}$ F such that $\mathsf{FloatLang}(F) = \mathsf{SufLang}(B)$.

- *Construction of E.* All we need to do, is to transform B in such a way that when in a monitored location at time t reading a TSS κ, it is always possible to continue a run on the suffix $[t, \infty]$ of κ from the monitored location. To achieve that specification, we construct E from B as follows:
  - *Locations.* We take $Q^E = Q^B \cup D$, where D is the following set of "dummy" locations: $D = \{q \mid q \in 2^{\mathsf{Limit}(\mathcal{P}^B \cup \mathcal{A}^B)}\}$. Thus D contains one element for each possible label. We will use the dummy locations to make possible the prolongation of every prefix of run that can reach a monitored location of B.
  - *Initial locations.* We take $Q_0^E = Q_0^B$, that is, the set of initial locations of E are the initial locations of B.
  - *Monitored locations.* We take $Q_M^E = Q_M^B$, that is, the set of monitored locations of E are the monitored locations of B.
  - *Transition relation.* The transition relation $\delta^E \subseteq Q^E \times Q^E$ is the union of the three following sets:
    1. $\{(q_1, q_2) \mid q_1, q_2 \in Q^B$ and $(q_1, q_2) \in \delta^B\}$. The moves possible in B are possible in E.
    2. $\{(q_1, q_2) \mid q_1 \in Q_M^B, q_2 \in D\}$. It is possible to move from a monitored location of B to all dummy locations.
    3. $\{(q_1, q_2) \mid q_1, q_2 \in D\}$. Within the dummy locations, the transition relation is not constraining. Note that it is not possible from a dummy location to get back to a location of B.
  - *Propositions and atomic clock constraints.* The propositions and atomic clock constraints used in E are the ones used in B: $\mathcal{P}^E = \mathcal{P}^B$, $\mathcal{A}^E = \mathcal{A}^B$.
  - *Labelling function.* The labeling function is defined as follows:
    * if $q \in Q^B$: $\lambda^E(q) = \lambda^B(q)$;
    * if $q \in D$: $\lambda^E(q) = q$.
  - *Accepting locations.* The set of accepting locations of E is simply the set of dummy locations: $Q_F^E = D$.

  E accepts as floating language all pairs (κ, t) such that κ allows a run to reach a monitored location of B at time t, that is $(\kappa, t) \in \mathsf{PreLang}(B)$.
- *Construction of F.* The construction also uses "dummy" locations and is very similar to the one for E, we leave it to the reader.

□

We are now equipped to prove the closure to complementation of $\mathsf{MEventClockTA}$:

Theorem 4.47 **($\mathsf{MEventClockTA}$-Complement)** *For every monitored event-clock automaton A, we can compute a monitored event-clock automaton B that accepts exactly the complement of the floating language of A, i.e.* $\mathsf{FloatLang}(B) = \overline{\mathsf{FloatLang}(A)}$.

Proof. By lemma 4.44, the floating language of A, where $Q_M^A = \{q_1, q_2, \ldots, q_n\}$ can be expressed as the union of the floating language of n single monitored location event-clock automata $A_1, A_2, \ldots, A_n$, i.e. $\mathsf{FloatLang}(A) = \bigcup_{i=1}^{i=n} \mathsf{FloatLang}(A_i)$. Also, note that $\overline{\mathsf{FloatLang}(A)} = \overline{\bigcup_{i=1}^{i=n} \mathsf{FloatLang}(A_i)}$ and thus $\overline{\mathsf{FloatLang}(A)} = \bigcap_{i=1}^{i=n} \overline{\mathsf{FloatLang}(A_i)}$. By lemma 4.46, we can compute $\overline{A_1}, \overline{A_2}, \ldots \overline{A_n}$, and by lemma 4.31, we can compute $B = \bigotimes_{i=1}^{i=n} \overline{A_i}$ that accepts the desired language. □

A direct corollary of the last theorem and the lemma about the equivalence between recursive event-clock automata and monitored recursive event-clock automata:

Corollary 4.48 **($\mathsf{REventClockTA}$-Complement)** *For every recursive event-clock automaton A, we can compute another recursive automaton B that accepts exactly the complement of the floating language of A, i.e. the pairs (κ, t) that are not accepted by A.* □

We now take a look at the complexity of this complementation procedure. This information will be used later in this section when we will characterize the complexity of decision problems for (monitored) recursive event-clock automata. We first define a notion of size for the (monitored) recursive event-clock automata.

Definition 4.49 **(Size of a $\mathsf{MEventClockTA}$)** We first define the notion of size for the base case, that is when the considered automaton A is a (monitored) floating automaton, we define the recursive case after.

- *Base case*: the size of a (monitored) floating automaton is characterized by:
  1. its number of locations $|Q^A|$, noted $\mathsf{NumLocs}(A)$;
  2. its number of possible labels $|2^{\mathsf{Limit}(\mathcal{P}^A)}|$, noted $\mathsf{NumAtomsSets}(A)$.
- *Recursive case*: the size of a (monitored) recursive event-clock automaton is characterized by:
  1. its number of locations $|Q^A|$, noted $\mathsf{NumLocs}(A)$;
  2. its number of possible labels $|2^{\mathsf{Limit}(\mathcal{P}^A)}|$, noted $\mathsf{NumAtomsSets}(A)$.
  3. the number of clock used by A, that is $|\{z_B \mid \exists (z_B \sim c) \in \mathcal{A}^A\}|$, this is noted $\mathsf{NumClocks}(A)$;
  4. the maximal constant that A use in its clock constraints, that is $\mathsf{Max}(\{c \mid \exists (z_B \sim c) \in \mathcal{A}^A\}|$, this is noted $\mathsf{MaxConst}(A)$;
  5. recursively, the size of its subautomata.

□

To ease the characterization of the size of the automaton obtained after applying the complementation procedure presented above, we use the figure 4.4.2. This figures schematizes the different step used in the complementation procedure. The following lemma characterizes the size of the automaton obtained after complementation:

Lemma 4.50 *For every monitored event-clock automaton A, we can compute a monitored event-clock automaton H that accepts exactly the complement of the floating language of A, i.e.* $\mathsf{FloatLang}(H) = \overline{\mathsf{FloatLang}(A)}$. *Further the size of H is defined in function of the size of A as follows:*

- *the number of locations of H is singly exponential in the number of locations of A, that is* $\mathsf{NumLocs}(H) = \mathcal{O}(2^{\mathsf{NumLocs}(A)})$;

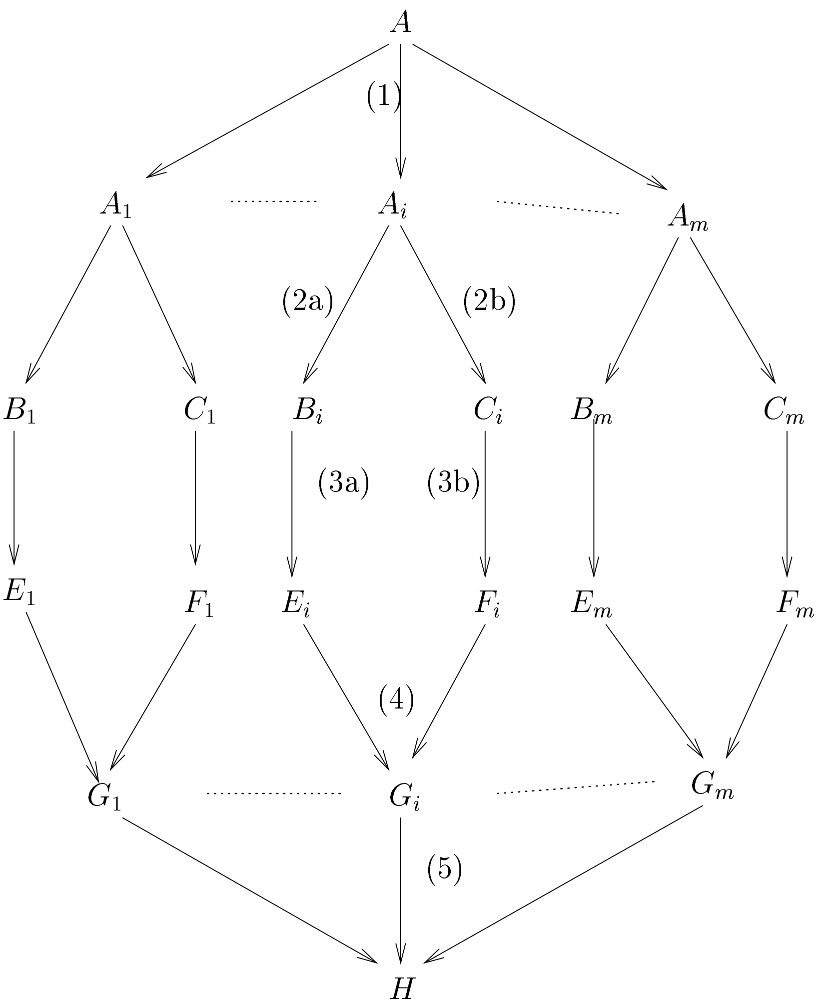

Figure 4: Complementation procedure for MEventClockTA.

- *the number of possible labels of H is equal to the number of possible labels of A, that is* $\mathsf{NumAtomsSets}(H) = \mathsf{NumAtomsSets}(A)$*;*
- *the maximal constant used by H in clock constraints is the same as the maximal constant used by A, that is* $\mathsf{MaxConst}(H) = \mathsf{MaxConst}(A)$*;*
- *the sizes of the subautomata of H are the same that sizes of the subautomata of A;*

Proof. We prove this lemma by inspecting the complexity of each transformations involved in the procedure for complementing the floating language of A, those transformations are depicted in figure 4.4.2.

- *Transformation (1).* The transformation simply change the set of monitored locations. The size of each A_i is equal to the size A;
- *Transformation (2a).* Each B_i is obtained from A_i by determinization. So we examine the determinization procedure, see proof of lemma 4.37. We first note that the step needed to obtain the non-repeating property can be neglected as its only effect is to multiply by 3 the number of locations, the other elements of the automaton remain unchanged. The subsets construction uses pairs composed of a set of locations of the non-repeating automaton as locations and labels of A. The labels part has no influence as the labels are the ones used by A. So the number of locations of each B_i is singly exponential in the number of locations of each A_i and thus singly exponential in the number of locations of A plus a dummy location is added for each label (to obtain a total automaton), so $\mathsf{NumLocs}(B_i) = \mathcal{O}(2^{\mathsf{NumLocs}(A)}) + \mathsf{NumAtomsSets}(A)$, the other elements of the automaton remains unchanged;
- *Transformation (2b).* Each C_i is obtained form A_i using the complementation procedure for Büchi automata which by theorem **??** leads to an exponential blow-up of the locations, that is $\mathsf{NumLocs}(C_i) = \mathcal{O}(2^{\mathsf{NumLocs}(A_i)})$ The other elements have the same size as in A.
- *Transformation (3a).* The transformation is described in the proof of lemma 4.39. Each automata E_i is obtained from B_i by adding a set of dummy locations. The number of such dummy locations is linear in the size of the number of possible labels for B_i which is equal to the number of possible labels for A. So the number of locations in each E_i is characterize as follows: $\mathsf{NumLocs}(E_i) = \mathcal{O}(2^{\mathsf{NumLocs}(A)} + \mathsf{NumAtomsSets}(A))$. The other elements of the size of E_i are as for A.
- *Transformation (3b).* This transformation is similar to the transformation (3a) and thus $\mathsf{NumLocs}(F_i) = \mathcal{O}(2^{\mathsf{NumLocs}(A)} + \mathsf{NumAtomsSets}(A))$.
- *Transformation (4).* The transformation consists in taking the union of the automata E_i and F_i. By theorem 4.29, as the two automaton E_i and F_i share the same set of possible labels, the number of locations of G_i is just the sum of the number of locations of E_i and F_i. So we have that $\mathsf{NumLocs}(G_i) = \mathcal{O}(2^{\mathsf{NumLocs}}(A) + \mathsf{NumAtomsSets}(A)) + \mathcal{O}(2^{\mathsf{NumLocs}(A)} + \mathsf{NumAtomsSets}(A))$ and thus $\mathsf{NumLocs}(G_i) = \mathcal{O}(2^{\mathsf{NumLocs}}(A) + \mathsf{NumAtomsSets}(A))$. The other elements of the size of G_i are as for A.
- *Transformation (5).* The transformation consists in taking the intersection of the m automaton G_i with $0 \leq i \leq m$, where m is the number of monitored locations in A and thus $m = \mathcal{O}(\mathsf{NumLocs}(A))$. By slightly generalizing the algorithm for intersection, which is defined for two $\mathsf{MEventClockTA}$ in the proof of theorem 4.31, and as G_i are defined on the same set of possible labels, we obtain that $\mathsf{NumLocs}(H) = \mathcal{O}(m \times (2^{\mathsf{NumLocs}(A)} + \mathsf{NumAtomsSets}(A)))$ and thus $\mathsf{NumLocs}(H) = \mathcal{O}(2^{\mathsf{NumLocs}(A)} + \mathcal{O}(\mathsf{NumLocs}(A)) \times \mathsf{NumAtomsSets}(A))$. The other elements of the size of H are as for A.

□

4.4.3 Closure under Partial Projection

Another important property of (monitored) recursive event-clock automaton is that they are *partially closed* under projection. Before proving this result, we need to introduce a new notion.

Definition 4.51 (FreeOfRTC) A *proposition p is not real-time constrained* into an monitored recursive event clock automaton A if this proposition does not appear in the set of propositions used by the subautomata of A. We note $\mathsf{FreeOfRTC}(A)$ the subset of propositions that are not real-time constrained by A. We define them formally as follows: $\mathsf{FreeOfRTC}(A) = \{q \in \mathcal{P}^A \mid \text{for all } B \in \mathsf{SUB}(A) : q \notin \mathcal{P}^B\}$. $\Box$

We can now state and prove the following theorem:

Theorem 4.52 (Partial Projection Closure) *For every monitored recursive event-clock automaton A defined on the set of propositions $\mathcal{P}$, for every subset of propositions $\mathcal{P}' \subseteq \mathcal{P}$ such that $\mathcal{P} \backslash \mathcal{P}' \subseteq \mathsf{FreeOfRTC}(A)$, we can construct a recursive event-clock automaton B that accepts the language* $\mathsf{FloatLang}(B) = \{(\kappa \downarrow \mathcal{P}', t) \mid (\kappa, t) \in \mathsf{FloatLang}(A)\}$.

Proof. We take B as A but change the labels as follows: for every locations q, $\lambda^B(q) = \lambda^A(q) \cap \mathsf{Limit}(\mathcal{P}^B \cup \mathcal{A}^B)$, that is, as $\mathcal{A}^B = \mathcal{A}^A$, we simply suppress the literals related to projected propositions. It is direct to show that B accepts the desired floating language. $\Box$

The constraint that imposes that projected propositions are only propositions that are not real-time constrained is essential. In fact, we will show later, that allowing projection of real-time constrained propositions strictly extends the expressive power of our recursive event-clock automata and would destroy their closure under negation. Again, we can derive the corresponding corollary for recursive event-clock automata.

4.5 Emptiness and Universality for Recursive Event-Clock Automata

We now show that the emptiness problem for a monitored recursive event clock automaton A, i.e. is the anchored language defined by the MEventClockTA A is empty or not, is decidable and we characterize the complexity of deciding this problem. We show that it is possible to reduce the emptiness problem for monitored recursive event-clock automata to the emptiness problem of non recursive automata for which a solution exists see [AFH94]. Again, the results for recursive event-clock automata are obtained as direct corollaries of the lemma 4.28 that states the equivalence between recursive event-clock automata and their monitored versions.

In the sequel, we show how to construct a propositional event-clock automaton that accepts TSS that are closely related to the TSS accepted by the recursive event-clock automaton. To define those TSS, we need some more ingredients.

For a MEventClockTA A with set of propositions $\mathcal{P}^A$, we construct a (non-recursive) EventClockTA B on the set of propositions

$$\begin{aligned}\mathcal{P}^B = &\mathcal{P}^A \\ &\cup \{p_C | C \in \mathsf{SUB}(A) \text{ or } C = A\} \\ &\cup \{p_{z_D \sim c} \mid \exists C \in \{A\} \cup \mathsf{SUB}(A) : (z_D \sim c) \in \mathcal{A}^C\}\end{aligned}$$

, i.e. we associate a new proposition to A and to each of its subautomata, moreover we associate a new proposition with each atomic real-time constraint appearing in A or in one of its subautomata.

In the sequel, we note

- $\mathcal{P}^{Aut}$ the set $\{p_C | C \in \mathsf{SUB}(A) \text{ or } C = A\}$
- and $\mathcal{P}^{Constr}$ the set $\{p_{z_D \sim c} \mid \exists C \in \{A\} \cup \mathsf{SUB}(A) : (z_D \sim c) \in \mathcal{A}^C\}$.

Further, the automaton B will use the following set of atomic clock constraints $\mathcal{A}^B = \{z_{p_D} \sim c \mid \exists p_{z_D \sim c} \in \mathcal{P}^{Constr}\}$. That is, we use a constraint $z_{p_D} \sim c$ over the proposition associated to the automaton D if there is a clock constraint $z_D \sim c$ over D in A or in one of its subautomata.

Definition 4.53 (Hintikka Property) Given a TSS κ, defined on $\mathcal{P}^A$, the $\mathcal{P}^B \setminus \mathcal{P}^A$ extension of κ, noted $\widehat{\kappa}$ defined on the set of propositions $\mathcal{P}^B$, has the timed Hintikka property for the $\mathsf{MEventClockTA}$ A if the following conditions are verified:

H1 $p_A \in \widehat{\kappa}(0)$, and for all time $t \in \mathbb{R}^+$:

H2 $p_{y_D \sim c} \in \widehat{\kappa}(t)$ iff there exists a time $t_1 > t$ such that:

- *either*: $p_D \in \widehat{\kappa}(t_1)$ and for all time $t_2 \in (t, t_1)$: $p_D \notin \widehat{\kappa}(t_2)$, and $t_1 - t \sim c$;
- *or*: for all time $t_2 > t_1$, there exists a time $t_3 \in (t_1, t_2)$ such that $p_D \in \widehat{\kappa}(t_3)$ and for all time $t_4 \in (t, t_1]$: $p_D \notin \widehat{\kappa}(t_3)$, and $(t_1 - t)^+ \sim c$.

H3 $p_{x_D \sim c} \in \widehat{\kappa}(t)$ iff there exists a time $t_1 \in [0, t)$ such that:

- *either*: $p_D \in \widehat{\kappa}(t_1)$ and for all time $t_2 \in (t_1, t)$: $p_D \notin \widehat{\kappa}(t_2)$, and $t - t_1 \sim c$;
- *or*: for all time $0 \leq t_2 < t_1$, there exists a time $t_3 \in (t_2, t_1)$ such that $p_D \in \widehat{\kappa}(t_3)$ and for all time $t_4 \in [t_1, t)$: $p_D \notin \widehat{\kappa}(t_3)$, and $(t - t_1)^+ \sim c$.

H4 if $p_C \in \widehat{\kappa}(t)$ then C has a t-monitored and accepted run on κ [3];

H5 if $p_C \notin \widehat{\kappa}(t)$ then C has no t-monitored and accepted run on κ.

Conditions H4 and H5 ensure that the proposition p_C associated to the automaton C, is true along $\widehat{\kappa}$ at time t iff C has a t-monitored and accepted timed run on $\widehat{\kappa}$. As a consequence, H1 imposes that A accepts $\widehat{\kappa}$ at time 0 and thus $\widehat{\kappa} \downarrow \mathcal{P}^A \in \mathsf{AncLang}(A)$, where $\widehat{\kappa} \downarrow \mathcal{P}^A$ denotes the TSS obtained from $\widehat{\kappa}$ by projecting propositions that are not in $\mathcal{P}^A$. H2 and H3 relates propositions $p_{z_D \sim c}$ to the semantics of the associated constraint $z_D \sim c$. In the sequel, we say that a TSS $\widehat{\kappa}$ that has the Hintikka property for A, is a timed Hintikka sequence for A, THS for short. □

The following lemma states how THS of an $\mathsf{MEventClockTA}$ A can help us to solve the emptiness problem of A:

Lemma 4.54 (Emptiness-Hintikka) *The anchored language of a monitored event-clock automaton A is non empty iff A has at least one timed Hintikka sequence.*

Proof. It is direct to show that if $\widehat{\kappa}$ is a Hintikka sequence for A then $\widehat{\kappa} \downarrow \mathcal{P}^A \in \mathsf{AncLang}(A)$. In fact, if $\widehat{\kappa}$ has the Hintikka sequence for A then $p_A \in \widehat{\kappa}(0)$ by condition H1 and then by H4, we know that A has a 0-monitored and accepted run on κ and thus $\kappa \in \mathsf{AncLang}(A)$. Now the extension $\widehat{\kappa}$ of $\kappa \in \mathsf{AncLang}(A)$ defined as follows:

$$\begin{aligned}\widehat{\kappa}(t) = &\kappa(t)\\ &\cup \{p_C \mid p_C \in \mathcal{P}^{Aut} \text{ and } \mathsf{Accept}_C(\kappa, t)\}\\ &\cup \{p_{z_D \sim c} \mid p_{z_D \sim c} \in \mathcal{P}^{Constr} \text{ and } (\kappa, t) \models z_D \sim c\}\end{aligned}$$

has the timed Hintikka property for A and is the unique extension of κ with this property. □

[3] Note that as $\widehat{\kappa}$ is an extension of κ, $\mathsf{Accept}_C(\kappa, t)$ iff $\mathsf{Accept}_C(\widehat{\kappa}, t)$.

In the sequel, we will show that the following lemma holds:

Lemma 4.55 (EventClockTA for Hintikka Sequences) *For every monitored recursive event-clock automaton A, we can construct a propositional event-clock automaton B that accepts exactly the timed Hintikka sequences of A, that is,* $\mathsf{AncLang}(B) = \{\kappa \mid \kappa \textit{ is a } \mathsf{THS} \textit{ of } A\}$.

Now we show that for each condition H1 to H5, we can construct a non recursive event-clock automaton that checks the condition. The final propositional event-clock automaton will simply be the product of the automata for each conditions, i.e. the automaton that accepts the intersection of the TSS accepted by each automaton. The construction that we will present is inspirated from the construction, proposed by Wolper et al to solve the satisfiability problem of the logic E-TL, see [WVS83, Wol83]. We now construct systematically an non recursive event-clock automaton for each timed Hintikka condition:

Automaton for condition H1,H2 and H3. We construct the EventClockTA $B_1 = (Q^{B_1}, Q_0^{B_1}, \delta^{B_1}, \mathcal{P}^{B_1}, \lambda^{B_1}, \mathcal{A}^{B_1}, Q_F^{B_1})$, as follows:

- *Propositions and atomic clock constraints* $\mathcal{P}^{B_1} = \mathcal{P}^B$ and $\mathcal{A}^{B_1} = \{z_{p_D} \sim c \mid p_{z_D \sim c} \in \mathcal{P}^B\}$, a clock is associated to each proposition associated to an automaton that appears in a clock constraint in A or one of its subautomata, those clocks will be used to enforce the right timing of those propositions;
- *Locations.* $Q^{B_1} = \{q \in 2^{\mathsf{Limit}(\mathcal{P}^B \cup \mathcal{A}^B)} \mid \forall p_{z_D \sim c} \in \mathcal{P}^{Constr} : p_{z_D \sim c} \in q \text{ iff } z_{p_D} \sim c \in q\}$. So B_1 contains a location for each possible label that respects the property that the proposition $p_{z_D \sim c}$ is in the label iff the corresponding constraint is also present. Intuitively, when $p_{z_D \sim c}$ is true along $\widehat{\kappa}$ at time t, it means that the constraint $z_D \sim c$ must be verified in (κ, t). As $p_D \in \widehat{\kappa}$ iff $(\kappa, t) \in \mathsf{FloatLang}(D)$ (by H4 and H5), we simply use the constraint $z_{p_D} \sim c$ to enforce the semantics of $z_D \sim c$. For example, if $y_D = 1$ is true in $(\widehat{\kappa}, t)$, it means that the following time $t_1 > t$ such that D accepts $\widehat{\kappa}$ must be $t_1 = t + 1$. We know by H4 and H5 that for all time $t \in \mathbb{R}^+$, $\mathsf{Accept}_D(\kappa, t)$ iff $p_D \in \widehat{\kappa}$, so we simply use the propositional clock y_{p_D} to enforce the semantics of $y_D = 1$: we check that $y_{p_D} = 1$ is verified.
- *Labeling function.* $\lambda^{B_1}(q) = q$, the labeling of location q is simply the literals that constitute the location;
- *Initial locations.* $Q_0^{B_1} = \{q \in Q^{B_1} | p_A \in q\}$, the initial condition impose that p_A is true initially;
- *Transition relation.* $\delta^{B_1} = \{(q_1, q_2) | q_1, q_2 \in Q^{B_1}\}$, i.e. there is no restriction on the transition relation;
- *Acceptance condition.* $Q_F^{B_1} = Q^{B_1}$, the accepting condition is trivial and thus does not impose any constraint on the accepted TSS.

The automaton B_1 ensures that p_A is true initially (by the definition of the initial locations and the definition of the labeling function) as we have defined the initial location as the locations labeled by p_A. Further, each time that the proposition $p_{x_D \sim c}$ ($p_{y_D \sim c}$, respectively) is true in a location, we decorate this location with the real-time constraint $x_{p_D} \sim c$ ($y_{p_D} \sim c$, respectively) which, by the semantics of clocks of EventClockTA, imposes the right timing on the last (first following, respectively) occurrence of p_D and by H4 and H5, ensures the verification of real-time constraints $x_D \sim c$ (resp.$y_D \sim c$) associated to the MEventClockTA D.

Automaton for condition H4 We construct an automaton $B_{2,C}$ for each $C \in A \cup \mathsf{SUB}(A)$. Basically, to enforce the property H4 for C, the automaton $B_{2,C}$ must, each time that it encounters a state in $\widehat{\kappa}$ where the proposition p_C is true, check that: "there exists a t-monitored run of C on $\widehat{\kappa}$". That can be done by checking the two following properties:

1. there exists a finite run of C that cover $\widehat{\kappa}$ for the interval $[0, t]$ and ends in a monitored location, say in q_m;

2. and that we can extend this run from q_m to cover the reminder of $\widehat{\kappa}$, i.e. the interval $[t, \infty)$, still respect the accepting condition of C.

The difficulty arises from the fact that we must check the existence of such runs each time that the proposition p_C is true, and the proposition p_C is potentially true in an infinite (and uncountable) numbers of time $t \in \mathbb{R}^+$. But fortunately, runs that are in the same location of C at a given time $t \in \mathbb{R}^+$ can be merged. In fact, as the value of clocks does not depend on the history of the run but only on the TSS the automaton is reading, two runs that reside in the same location have the same possible futures. More precisely, if $\rho_1^{[0,t]}$ and $\rho_2^{[0,t]}$ are two prefixes of runs on TSS $\widehat{\kappa}$, such that $\rho_1^{[0,t]}(t) = \rho_2^{[0,t]}(t)$ then if $\rho_1^{[0,t]} \cdot \rho_3^{(t,\infty)}$ is a accepted run of A on κ then so is $\rho_2^{[0,t]} \cdot \rho_3^{(t,\infty)}$. Note that this property is not true for timed automata in general. In fact, in a timed automaton run the value of clocks at a given time t depends on the history of the run up to that time t. So two prefixes of runs that at time t end up in the same location do not necessarily have the same futures, as their clock values can be different. This technique is again inspirated by the decision procedure for E-TL.

Let us now show in details how we can solve the problem. To simplify the presentation, we first define two transition structures.

Definition 4.56 (Transition Structure) A *transition structure* is a four-tuple $\Lambda = (S, S_0, R, F)$ where:

- S is a set of states;
- $S_0 \subseteq S$ is a set of initial states;
- $R \subseteq S \times S$ is the transition relation;
- and either $F \subseteq S$ is a set of accepting states, or $F \subseteq 2^S$ is a set of sets of accepting states. We will use set of accepting states when we will need to define a Büchi acceptance condition and we will use set of sets of accepting states when we will need to define a generalized Büchi condition.

We will use transition structure as intermediate objects.

We construct one transition structure for the two properties above and define how to take their product in order to obtain the automaton B_2 that checks condition H4.

- *Transition structure* Λ^1. To check that there exists a run on the prefix of $\widehat{\kappa}$ up to time t with $p_C \in \widehat{\kappa}(t)$, we simply maintain a deterministic version of C, see definition 4.35 and lemma 4.37. We note this deterministic version D and the structure $\Lambda^1 = (S^1, S_0^1, R^1, F^1)$ is defined from D as follows:
  - $S^1 = Q^D$, that is, the states of the transition structure Λ^1 is the set of locations of the deterministic version of C, and thus a state of Λ^1 is a set of locations of C;
  - $S_0^1 = Q_0^D$, the initial states of Λ^1 are the initial locations of D;
  - $R = \delta^D$, the transition relation is as in D;
  - $F = S^1$, each state is accepting and thus the accepting condition is trivial.

  This deterministic structure tells us at each moment, when reading $\widehat{\kappa}$, in which locations of C the control can reside. As a consequence, the automaton tells us in which locations all possible runs can be. We will use this information in order to start runs for suffixes only from monitored locations where the control can reside.
- *Transition structure* Λ^2. To check the existence of runs on the suffixes of $\widehat{\kappa}$ from time t, we consider a transition structure $\Lambda^2 = (S^2, S_0^2, R^2, F^2)$, where:
  - the states of Λ^2 are n-tuples of locations $\langle l_1, l_2, \ldots, l_n \rangle$ of C, thus $n = |Q^C|$. n-tuples are sufficient because, at each moment, the control of the automaton C can be, at most, in n different locations and we do not need more because we are allowed to merge runs that are in the same location. Each l_i belongs to $Q^C \cup \{\perp\}$, the special value $\perp$ is used for l_i when there is no active i^{th} run. We further impose the following properties to the tuples: $\langle l_1, \ldots, l_n \rangle \in S^2$ iff there exists a j, $1 \leq j \leq n+1$, such that:

1. for all k, $j \leq k \leq n$: $l_k = \bot$;
2. for all k, $1 \leq k < j$: $l_k \in Q^C$;
3. for all k_1, k_2, $1 \leq k_1 < k_2 < j$: $l_{k_1} \neq l_{k_2}$;
4. for all k_1, k_2, $1 \leq k_1 < k_2 < j$: $\lambda^C(l_{k_1}) = \lambda^C(l_{k_2})$.

The conjunction of condition (1) and (2) ensures that "real" locations occupy the first places in the tuple. Condition (3) imposes that all locations are different in the tuple. This is necessary as we have only n places and we must check potentially infinitely many runs, therefore, we must merge runs that reach the same location. Finally, in (4) we require that locations in the tuples have the same label. In fact, at each time t of a TSS, only one label is true so, at each time t, the control of C can only be in locations that share the same label. In the sequel, we use the notation $\lambda^C(\langle l_1, l_2, \ldots, l_n \rangle)$ to refer to that label.

- As initial states of Λ^2, we take $S_0^2 = S^2$.
- Let us now define *the transition relation* of the structure Λ^2: we have $(\langle l_1^1, l_2^1, \ldots, l_n^1 \rangle, \langle l_1^2, l_2^2, \ldots, l_n^2 \rangle) \in R^2$ iff for all k, $1 \leq k \leq n$: if $l_k^1 \neq \bot$ then there exists j, $1 \leq j \leq k$ and $(l_k^1, l_j^2) \in \delta^C$. That is each (non dummy) location of the first tuple has a δ^C-successor location in the second tuple, merging runs is allowed as j can be strictly less than k.
- Let us now expose how we can check that each run simulated in the structure Λ^2 respects the acceptance condition of C. To solve this problem we use a generalized Büchi acceptance condition: we define n sets of accepting locations, a run will be accepting if it has, for each n sets infinitely many positions in the set. The sets are defined as follows:
$$F_i = \{\langle l_0, l_1, \ldots, l_n \rangle | \text{ either } l_i = \bot \text{ or } l_i \in Q_F^C\}$$
In the sequel, we note $\Lambda^2.F_i$, the i^{th} set of accepting states of the transition structure Λ^2. Let us show that this choice for the accepting condition is correct. Consider a run that starts in the i^{th} coordinate of the tuples. Either this run is merged with another run $j < i$. In that case $l_i = \bot$ until we start another run, or $l_i = \bot$ for ever, in the last case, the run is accepted. Now, if the run continues for ever in a coordinate $k \leq j$, which must arrive sooner or later, then we must check that the run goes infinitely often through an accepting location of C which is checked by the set $\Lambda^2.F.F_k$.

We are now in position to define the non recursive automaton $B_{2,C} = (Q^{B_{2,C}}, Q_0^{B_{2,C}}, \delta^{B_{2,C}}, \mathcal{P}^{B_{2,C}}, \lambda^{B_{2,C}}, \mathcal{A}^{B_{2,C}}, Q_F^{B_{2,C}})$:

- *Locations.* $Q^{B_{2,C}}$ is the set of tuples $\langle \lambda, s_1, s_2 \rangle$ where:
  - $\lambda \in 2^{\mathsf{Limit}(\mathcal{P}^B \cup \mathcal{A}^B)}$, λ is a label;
  - $s_1 \in S^1$, this part will be used to check the constraints over prefixes as explained above;
  - $s_2 \in S^2$, this part will be used to check the constraints over suffixes as explained above;

  that respect the following restrictions (with $s_2 = \langle l_1, \ldots, l_n \rangle$):
  1. (a) for all $p \in \mathcal{P}^C$: $p \in \lambda$ iff $p \in \lambda^C(s_1)$ iff $p \in \lambda^C(\langle l_1, \ldots, l_n \rangle)$;
     (b) for all $z_D \sim c \in \mathcal{A}^C$: $p_{z_D \sim c} \in \lambda$ iff $(z_D \sim c) \in \lambda^C(s_1)$ iff $(z_D \sim c) \in \lambda(\langle l_1, \ldots, l_n \rangle)$.
  2. if $p_C \in \lambda$ and $s_2 = \langle l_1, \ldots, l_n \rangle$ then there exists j, $1 \leq j \leq n$, such that $l_j \in Q_M^C$; that is, if p_C is true then it is necessary to check that there exists a run of C on the rest of the TSS that starts in a monitored location, the structure Λ^2 will check for the existence of such a run;
  3. if $s_2 = \langle l_1, l_2, \ldots, l_n \rangle$ then for all j such that $1 \leq j \leq n$ and $l_j \neq \bot$, we have $l_j \in s_1$; this constraint imposes that the locations active in runs are a subset of the locations where the control of the automaton can reside (information given by the structure Λ^1).
- *Initial locations.* $Q_0^{B_{2,C}} = \{(\lambda, s_1, s_2) | s_1 \in S_0^1\}$, recall that S_0^1 contains all the sets of locations where the automaton C can start a run;
- *Transition Relation.* $[(\lambda^1, s_1^1, s_2^1), (\lambda^2, s_1^2, s_2^2)] \in \delta^{B_{2,C}}$ iff $(s_1^1, s_1^2) \in R^1$ and $(s_2^1, s_2^2) \in R^2$, thus there is a transition in $B_{2,C}$ if the transition is possible in both Λ^1 and Λ^2;

- *Propositions and atomic clock constraints.* The propositions and the clocks constraints are as for B: $\mathcal{P}^{B_{2,C}} = \mathcal{P}^B$ and $\mathcal{A}^{B_{2,C}} = \mathcal{A}^B$.
- *Labeling function.* For all $(\lambda, s_1, s_2) \in Q^{B_2}$, $\lambda^{B_2}((\lambda, s_1, s_2)) = \lambda$.
- *Accepting condition.* For the acceptance condition, we transpose into $B_{2,C}$ the constraints of Λ^2. So we use the following generalized Büchi acceptance condition: $Q_F^{B_{2,C}} = \{F_1, F_2, \ldots, F_n\}$ where each F_i is defined by $\{(\lambda, s_1, \langle l_0, \ldots, l_i, \ldots, l_n\rangle) \mid l_i \in Q_F^C \vee l_i = \bot\}$.

Now, B_2 is obtained by taking the product of each $B_{2,C}$ for $C \in \{A\} \cup \mathsf{SUB}(A)$.

Automaton for condition H5 One way to solve this problem would be to consider for each automaton $C \in \{A \cup \mathsf{SUB}(A)\}$, its complement $\overline{C}$ and check condition H4 for that automaton. As we have proved that $\mathsf{MEventClockTA}$ are closed under complementation, this strategy works to complete our construction for the emptiness problem of $\mathsf{MEventClockTA}$. But this method does not match the optimal complexity since after complementation, which costs an exponential, we should still construct the deterministic structure (Λ^1) and the tuple-structure (Λ^2) which also costs one exponential. Applying this simple idea would result in a doubly exponential blow-up in the number of locations of the constructed automaton giving an EX-PSPACE procedure. It is possible to solve the problem with only one exponential, yielding a PSPACE procedure, with the following idea (again, adapted from [SVW85]): for each automaton C, we construct an automaton $B_{3,C}$ that enforces exactly the negation of H5 for each C, that is "there exists a time $t \in \mathbb{R}^+$ such that $p_C \notin \widehat{\kappa}(t)$ and C has a t-monitored and accepted run on $\widehat{\kappa}$". After, we take the union of all those automata and complement this union, we obtain a single automaton B_3 that checks H5 for each automaton $C \in \{A \cup \mathsf{SUB}(A)\}$. The construction is singly exponential (the one that occurs during the complementation). Let us now show how to construct the automaton $B_{3,C}$. The idea behind the construction is the following: we construct an automaton which is essentially the product of C with a simple transition structure Λ that ensures, when we take the product between C and Λ, that p_C is eventually false and at the same time C is in a monitored location. The structure Λ is defined as follows:

- *States.*The set of states S is the set of 3-tuples (i, ξ_1, ξ_2) such that $i \in \{1, 2, 3\}$, $\xi_1 \in \{M, \bar{M}\}$, $\xi_2 \in \{p_C, \bar{p_C}\}$, with the restriction that if $i = 2$ then $\xi_1 = M$ and $\xi_2 = \bar{p_C}$. The intuition is that when the structure Λ is in a state tagged by 2 then C is in a monitored location and the proposition p_C is false. We will use the initial condition, transition relation and acceptance condition to ensure that each run of Λ eventually passes through a state tagged with 2.
- *Initial states* are $S_0 = \{(i, \xi_1, \xi_2) \in Q | i \in \{1, 2\}\}$. Initially, the control can only be into part 1 or part 2 of the structure.
- *Transitions*: $((i^1, \xi_1^1, \xi_2^1), (i^2, \xi_1^2, \xi_2^2)) \in R$ iff either $i^2 = i^1$ or $i^2 = i^1 + 1$. The control of the automaton can only go from part 1 to part 2 and then to part 3. Consequently, when in part 1, the control must cross part 2 to attain the accepting locations.
- *Acceptance*: $F = \{(i, \xi_1, \xi_2) | i \in \{2, 3\}\}$, the accepting states are those tagged with 2 or 3.

We now construct $B_{3,C}$ from C and $\Lambda = (S, S_0, R, F)$ as follows:

- *Locations.* $Q^{B_{3,C}}$ is the set of 3-tuples (s, q, λ) such that:
  - $s \in S$;
  - $q \in Q^C$;
  - $\lambda \in 2^{\mathsf{Limit}(\mathcal{P}^B \cup \mathcal{A}^B)}$;
  - if $s = (i, \xi_1, \xi_2)$ then $\xi_1 = M$ iff $q_2 \in Q_M^C$, that is the control is in a M-state of Λ iff the control is in a monitored location in Q^C.
- *Initial locations.* The set of initial locations $Q_0^{B_{3,C}} = \{(s, q, \lambda) \mid s \in S_0 \text{ and } q \in Q_0^C\}$, that is, we check that the structure Λ and the automaton C respects their initial requirement;
- *Transition relation.* We have $[(s_1, q_1, \lambda_1), (s_2, q_2, \lambda_2)] \in \delta^{B_{3,C}}$ iff
  1. $(s_1, s_2) \in R$;

2. $(q_1, q_2) \in \delta^{B_{3,C}}$ or $q_1 = q_2$;

So, we check the transition relation of both Λ and C (stuttering steps are allowed in C).

- *Propositions and atomic clock constraints.* The propositions and atomic clock constraints are as in automaton B: $\mathcal{P}^{B_{3,C}} = \mathcal{P}^B$, $\mathcal{A}^{B_{3,C}} = \mathcal{A}^B$;
- *Labeling function.* The labeling function is defined as follows: $\lambda^{B_{3,C}}((s, q, \lambda)) = \lambda$.
- *Accepting locations.* The accepting condition is defined as follows: to be accepted, a run must respect the conjunction of the accepting conditions for C and the transition structure Λ. Therefore, we define the following acceptance condition: $Q^{B_{3,C}} = \{F_1, F_2\}$ with $F_1 = \{(s, q, \lambda) \mid s \in \Lambda.F\}$ and $F_2 = \{(s, q, \lambda) \mid q \in Q^C_F\}$.

To obtain the automaton B_2, we just complement the union of the set of automata $\{B_{3,C} \mid C \in A \cup \mathsf{SUB}(A)\}$.

We finally obtain the non recursive event-clock automaton B by taking the product of the automata B_1, B_2, B_3.

The following theorem follows from the previous construction.

Theorem 4.57 (REventClockTA-Emptiness) *The emptiness problem for recursive event-clock automata is* PSPACE-COMPLETE.

To check the universality problem, we use the same construction with H1 replaced by:

H1' $p_A \notin \widehat{\kappa}(0)$

and check that the language of the constructed propositional event-clock automaton is empty, so we have:

Theorem 4.58 (REventClockTA-Universality) *The universality problem for recursive event-clock automata is* PSPACE-COMPLETE.

4.6 Expressiveness: EventClockTL $\subset$ REventClockTA

In section 3.3.2, we have shown that propositional (non recursive) event-clock automata are not sufficiently expressive to define all EventClockTL-properties. In this section, we show that, on the contrary, REventClockTA are sufficiently expressive to define all EventClockTL-properties. We first introduce some new notions.

Definition 4.59 (level of EventClockTL formulas) The level of an EventClockTL formula ϕ is computed by the following recursive function level:

- $\mathsf{level}(p) = 0$;
- $\mathsf{level}(\phi_1 \vee \phi_2) = \mathsf{Maximum}(\mathsf{level}(\phi_1), \mathsf{level}(\phi_2))$;
- $\mathsf{level}(\neg\phi_1) = \mathsf{level}(\phi_1)$;
- $\mathsf{level}(\phi_1 \mathcal{U} \phi_2) = \mathsf{Maximum}(\mathsf{level}(\phi_1), \mathsf{level}(\phi_2))$;
- $\mathsf{level}(\phi_1 \mathcal{S} \phi_2) = \mathsf{Maximum}(\mathsf{level}(\phi_1), \mathsf{level}(\phi_2))$;
- $\mathsf{level}(\rhd_I \phi_1) = 1 + \mathsf{level}(\phi_1)$;
- $\mathsf{level}(\lhd_I \phi_1) = 1 + \mathsf{level}(\phi_1)$;

That is the *level* of a formula ϕ is the number of imbrications of real-time operators in ϕ. $\Box$

We say that "ϕ is a level_i formula" if $\mathsf{level}(\phi) = i$. In the following proofs, we will reason by induction on the structure of level_i formulas, we define the grammar corresponding to those formulas:

Definition 4.60 (Grammar of level_i-formulas) The following grammar rule define the level_0 EventClockTL formulas:

$$\phi ::= p \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{S} \phi_2$$
$$\text{where } \phi_1 \text{ and } \phi_2 \text{ are } \mathsf{level}_0 \text{ formulas.}$$

Note that level_0 formulas are LTR formulas. Recursively, the following grammar rule define the level_i EventClockTL formulas:

$$\phi ::= p \mid \rhd_I \phi_3 \mid \lhd_I \phi_3 \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{S} \phi_2$$
$$\text{where } \phi_1 \text{ and } \phi_2 \text{ are } \mathsf{level}_j \text{ formulas where } 0 \leq j \leq i \text{ and } \phi_3 \text{ is a } \mathsf{level}_k \text{ formula where } 0 \leq k < i.$$

□

For example, $\rhd_{=1} \rhd_{=1} p$ is a level_2 formula.

We define the following slightly non-classical notion of *closure* of a formula:

Definition 4.61 (Closure Set) Let ϕ be an EventClockTL formula, we define the closure of ϕ, with the help of the recursive function Cl:

- $\mathsf{Cl}(p) = \{p\}$;
- $\mathsf{Cl}(\phi_1 \vee \phi_2) = \mathsf{Cl}(\phi_1) \cup \mathsf{Cl}(\phi_2) \cup \{\phi_1 \vee \phi_2\}$;
- $\mathsf{Cl}(\neg\phi_1) = \mathsf{Cl}(\phi_1)$;
- $\mathsf{Cl}(\phi_1 \mathcal{U} \phi_2) = \mathsf{Cl}(\phi_1) \cup \mathsf{Cl}(\phi_2) \cup \{\phi_1 \mathcal{U} \phi_2\}$;
- $\mathsf{Cl}(\phi_1 \mathcal{S} \phi_2) = \mathsf{Cl}(\phi_1) \cup \mathsf{Cl}(\phi_2) \cup \{\phi_1 \mathcal{S} \phi_2\}$;
- $\mathsf{Cl}(\rhd_I \phi_1) = \{\rhd_I \phi_1\}$;
- $\mathsf{Cl}(\lhd_I \phi_1) = \{\lhd_I \phi_1\}$;

The *closure of the formula* ϕ, denoted $\overline{\mathsf{Cl}}(\phi)$, is the set $\mathsf{Cl}(\phi)$ closed by negation, that is $\overline{\mathsf{Cl}}(\phi) = \{\psi, \neg\psi \mid \psi \in \mathsf{Cl}(\phi)\}$. □

In that non-classical notion of closure, the real-time subformulas $\rhd_I \phi_3$ and $\lhd_I \phi_3$ are considered as atomic formulas. Let us now consider the following lemma:

Lemma 4.62 (EventClockTL − Fine TSS) *For every set of propositions $\mathcal{P}$, for every* TSS *κ, if κ is* $\mathsf{Limit}(\mathcal{P})$ − Fine *and alternating, then κ is also ϕ* − Fine *for every* level_0-EventClockTL *formula ϕ whose propositions are in $\mathcal{P}$.*

Proof. We prove this lemma by induction of the structure of level_0-formulas.

- *Base case.* If $\phi = p$ with $p \in \mathcal{P}$ then the lemma is trivially verified as $p \in \mathsf{Limit}(\mathcal{P})$.
- *Induction case.* The induction hypothesis tell us that for ϕ_1 and ϕ_2 which are level_0 formulas, we know that κ is ϕ_1 − Fine as well as ϕ_2 − Fine. Let us also observe that a singular interval can not be refined. So we only have to show that level_0 formulas have a constant truth value in all open intervals of κ. Now let us treat each construction of the grammar:
  - let $\psi = \phi_1 \vee \phi_2$. Let us consider the open interval I_i. There are four possible cases: ϕ_1 and ϕ_2 are constantly true during I_i, ϕ_1 is constantly true during I_i and ϕ_2 is constantly false, ... Let us treat the first case as an example, the other cases are treated similarly. If ϕ_1 and ϕ_2 are constantly true during I_i then by the semantics of the $\vee$-operator, $\phi_1 \vee \phi_2$ is constantly true during I_i. Thus the sequence of intervals does not need to be refined.
  - let $\psi = \neg\phi_1$. In that case, if ϕ_1 is constantly true during I_i then ψ is constantly during this interval, and conversely. Again, the sequence of intervals does not need to be refined.

- let $\psi = \phi_1 \mathcal{U} \phi_2$. To treat that case, let us make the hypothesis that $(\kappa, t) \models \phi_1 \mathcal{U} \phi_2$ for some $t \in I_i$. We will show that this implies that for all time $t_1 \in I_i$, $(\kappa, t_1) \models \phi_1 \mathcal{U} \phi_2$. We will treat the negation after. By the semantics of the $\mathcal{U}$-operator, we know that: there exists a time $t' > t$ such that $(\kappa, t) \models \phi_2$ and for all time $t'' \in (t, t')$, $(\kappa, t'') \models \phi_1 \vee \phi_2$. Let us first make the hypothesis that t' belongs to the interval I_i. By induction hypothesis, we know that for all time $t_1 \in I_i$, $(\kappa, t_1) \models \phi_2$. As I_i is open, it is easy to see that $(\kappa, t_1) \models \phi_1 \mathcal{U} \phi_2$ for all $t_1 \in I_i$. Now let us make the conserve hypothesis, the first time where ϕ_2 holds is not in I_i but after. By induction hypothesis, this implies that for all time $t_1 \in I_i$, $(\kappa, t_1) \not\models \phi_2$. By semantics of the $\mathcal{U}$-operator, we know that ϕ_1 must be true just after t within I_i. By induction hypothesis ϕ_1 is then constantly true within I_i and thus also $\phi_1 \mathcal{U} \phi_2$. Let us now turn to the case where there is a time $t \in I_i$ where $(\kappa, t) \not\models \phi_1 \mathcal{U} \phi_2$. We already prove that if there exists a time $t' \in I_i$ such that $(\kappa, t') \models \phi_1 \mathcal{U} \phi_2$, there does not exists a time $t'' \in I_i$ such $(\kappa, t'') \not\models \phi_1 \mathcal{U} \phi_2$. Thus as $(\kappa, t) \not\models \phi_1 \mathcal{U} \phi_2$ holds, we know that there can not exists such a t'.
- let $\psi = \phi_1 \mathcal{S} \phi_2$. This case is treated in the same way that the $\mathcal{U}$-case and is left to the reader.

□

This lemma will allow us, in the next proof, to tag locations of monitored event-clock automata with formulas of the logic and still keep the property that the control can only resides in a location for a singular interval of time only if the label of that location is singular.

Lemma 4.63 (EventClockTL $\subseteq$ MEventClockTA) *For every* EventClockTL *formula ϕ we can construct a* MEventClockTA *A_ϕ that accepts exactly the pairs (κ, t), where κ is defined on the set of propositions $\mathcal{P}$ appearing in ϕ and $t \in \mathbb{R}^+$, such that $(\kappa, t) \models \phi$.*

Proof. To establish this result, we reason by induction on the level of formulas.

- *Base case.* Let consider ϕ a level_0-formula. We first define a transition structure $\Lambda = (S, S_0, R, F)$ that checks the semantics of the propositional and temporal operators of level_0-formula. After, we will transform this structure into an monitored floating automaton. We define the elements of Λ as follows:
  - *States.* S is the set of pairs (a, ς) where $a \in 2^{\overline{\mathsf{Cl}}(\phi)}$ with $\top \in a$, $\varsigma \in \{\mathsf{open}, \mathsf{sing}\}$ (indicating if the control can stay in the state for an open interval of time or just a singular interval of time) and the following properties are verified:
    1. for all $\phi_1 \in \overline{\mathsf{Cl}}(\phi)$: $\phi_1 \in a$ iff $\neg\phi_1 \notin a$;
    2. for all $(\phi_1 \vee \phi_2) \in \overline{\mathsf{Cl}}(\phi)$: $\phi_1 \vee \phi_2 \in a$ iff $\phi_1 \in a$ or $\phi_2 \in a$;
    3. for all $(\phi_1 \mathcal{U} \phi_2) \in \overline{\mathsf{Cl}}(\phi)$:

       3.a if $\phi_2 \in a$ and $\varsigma = \mathsf{open}$ then $\phi_1 \mathcal{U} \phi_2 \in a$;

       3.b if $\phi_1 \mathcal{U} \phi_2 \in a$ and $\varsigma = \mathsf{open}$ then $\phi_1 \in a$ or $\phi_2 \in a$;
    4. for all $(\phi_1 \mathcal{S} \phi_2) \in \overline{\mathsf{Cl}}(\phi)$:

       a if $\phi_2 \in a$ and $\varsigma = \mathsf{open}$ then $\phi_1 \mathcal{S} \phi_2 \in a$;

       b if $\phi_1 \mathcal{S} \phi_2 \in a$ and $\varsigma = \mathsf{open}$ then $\phi_1 \in a$ or $\phi_2 \in a$;

    (1) and (2) enforces the semantics of propositional operators; (3.a) and (3.b) enforces local consistency for the until operator; (4.a) and (4.b) are the local consistency rules for the since operator.
  - *Initial states.* The set of initial states is the subset of pairs $(a, \varsigma) \in S$ such that $\varsigma = \mathsf{sing}$ and there does not exist $\phi_1 \mathcal{S} \phi_2 \in \bar{\mathsf{Cl}}(\phi)$ and $\phi_1 \mathcal{S} \phi_2 \in a$. That is an initial state is singular and it does not contains a since formula in positive form.
  - *Transition relation.* The transition relation R is the subset $[(a_1, \varsigma_1), (a_2, \varsigma_2)]$ of $S \times S$ that respects the following restrictions:
    1. $\varsigma_1 = \mathsf{open}$ and $\varsigma_2 = \mathsf{sing}$, or, $\varsigma_1 = \mathsf{sing}$ and $\varsigma_2 = \mathsf{open}$;

2. The following rules express how until formulas are transfered from one state to the next of the transition structure:
   2.a $\phi_1 \mathcal{U} \phi_2 \in a_1 \wedge \varsigma_1 = \mathsf{sing}$ iff $\phi_1 \mathcal{U} \phi_2 \in a_2$;
   2.b $\phi_1 \mathcal{U} \phi_2 \in a_1 \wedge \varsigma = \mathsf{open} \wedge \phi_2 \notin a_1$, implies $(\phi_1 \mathcal{U} \phi_2 \in a_2 \wedge \phi_1 \in a_2) \vee \phi_2 \in a_2$;
   2.c $\phi_1 \in a_1 \wedge \varsigma_1 = \mathsf{open} \wedge (\phi_1 \in a_2 \vee (\phi_2 \in a_2 \wedge \phi_1 \mathcal{U} \phi_2 \in a_2))$ implies $\phi_1 \mathcal{U} \phi_2 \in a_1$.
3. The following are for the since formulas:
   3.a $\phi_1 \mathcal{S} \phi_2 \in a_2 \wedge \varsigma = \mathsf{sing}$ iff $\phi_1 \mathcal{S} \phi_2 \in a_1$;
   3.b $\phi_1 \mathcal{S} \phi_2 \in a_2 \wedge \varsigma_2 = \mathsf{open} \wedge \phi_2 \notin a_2$ implies $\phi_2 \in a_1 \vee (\phi_1 \in a_1 \wedge (\phi_1 \mathcal{S} \phi_2) \in a_1)$;
   3.c $\phi_1 \in a_2 \wedge \varsigma_2 = \mathsf{open} \wedge (\phi_2 \in a_1 \vee \phi_1 \mathcal{S} \phi_2 \in a_1)$ implies $\phi_1 \mathcal{S} \phi_2 \in a_2$

– *Accepting states.* As usual, we use a generalized Büchi acceptance condition. For each formula $\phi_1 \mathcal{U} \phi_2 \in \overline{\mathsf{Cl}}(\phi)$, there is a set $\Lambda.F.F_{\phi_1 \mathcal{U} \phi_2} = \{(a, \varsigma) \mid \phi_1 \mathcal{U} \phi_2 \notin a \vee \phi_2 \in a\}$.

We are now equipped to define the monitored floating automaton A_ϕ. We construct $A_\phi = (Q^{A_\phi}, Q_0^{A_\phi}, Q_M^{A_\phi}, \delta^{A_\phi}, \mathcal{P}^{A_\phi}, \lambda^{A_\phi}, Q_{\mathsf{F}}^{A_\phi})$ as follows:

– *Locations.* The set of locations Q^{A_ϕ} is the set of pairs $((a, \varsigma), \chi)$ such that:
  1. $(a, \varsigma) \in S$;
  2. χ is a label that is open if and only if $\varsigma = \mathsf{open}$;
  3. the labeling is propositionally consistent with the formula in a: for all proposition $p \in \mathcal{P}$: $p \in \chi$ iff $p \in a$.
– *Initial locations.* The set of initial locations $Q_0^{A_\phi}$ is the subset of locations $((a, \varsigma), \chi) \in Q^{A_\phi}$ such that $(a, \varsigma) \in S_0$;
– *Monitored locations.* The set $Q_M^{A_\phi}$ of monitored locations is the subset of locations $((a, \varsigma), \chi) \in Q^{A_\phi}$ such that $\phi \in a$, that is the subset of locations where the formula ϕ is true;
– *Transition relation.* The transition relation is the set of pairs $[((a_1, \varsigma_1), \chi_1), ((a_2, \varsigma_2), \chi_2)]$ with $((a_i, \varsigma_i), \chi_i) \in Q^{A_\phi}$ for $i \in \{1, 2\}$, such that: $[(a_1, \varsigma_1), (a_2, \varsigma_2)] \in R$;
– *Propositions.* The set of propositions used by A_ϕ is the set of propositions that appear in the formula ϕ, i.e. $\mathcal{P}^{A_\phi} = \{p \mid p \in \overline{\mathsf{Cl}}(\phi)\}$;
– *Labeling function.* The labeling function λ^{A_ϕ} is defined as follows: $\lambda^{A_\phi}(((a, \varsigma), \chi)) = \chi$;
– *Accepting locations.* We transfer in A_ϕ the generalized Büchi acceptance condition of the transition structure $\Lambda : Q_F^{A_\phi}$ is the set of sets of accepting locations $\{F_1, \dots, F_n\}$ where each F_i corresponds to a set of accepting states in S as follows: $F_i = \{((a, \varsigma), \chi) \mid (a, \varsigma) \in \Lambda.F.F_i\}$.

It is routine to prove that the constructed automaton A_ϕ accepts the right floating language.

- *Induction case.* By induction hypothesis, we know that for each formula ψ of level_j with $j < i$, we are able to construct a congruent monitored recursive event-clock automaton A_ψ. Let us show that we can construct a automaton for each formula of level_i. By inspecting the grammar rules for level_i-formulas, it is not difficult to see that if we consider real-time formulas as atomic, the level_i-formulas are constructed in the same way as level_0-formulas. The construction of A_ϕ will be exactly as for the base case with the exception that we must treat the real-time formulas. We treat them as follows: for each formula $\triangleleft_I \phi_3$, we use the (history) atomic real-time constraint $x_{A_{\phi_3}} \in I$, and for each formula $\triangleright_I \phi_3$, we use the (predicting) atomic real-time constraint $y_{A_{\phi_3}} \in I$. Those constraints have the property, by induction hypothesis, that: for every TSS κ, every time $t \in \mathbb{R}^+$: $(\kappa, t) \models \triangleleft_I \phi_3$ iff $(\kappa, t) \models x_{A_{\phi_3}} \in I$ and $(\kappa, t) \models \triangleright_I \phi_3$ iff $(\kappa, t) \models y_{A_{\phi_3}} \in I$. Finally, when constructing the automaton A_ϕ, we use as set of propositions $\mathcal{P}^{A_\phi}$ the set of propositions that appears into formula ϕ and we check that the following additional rule for locations: if $((a, \varsigma), \chi) \in Q^{A_\phi}$ then for each real-time constraints $\triangleleft_I \phi_1$, $\triangleleft_I \phi_1 \in a$ iff $(x_{A_{\phi_1}} \in I) \in \chi$, and similarly for the future real-time operators: for each real-time constraints $\triangleright_I \phi_1$, $\triangleright_I \phi_1 \in a$ iff $(y_{A_{\phi_1}} \in I) \in \chi$. Again, it is routine to prove that the constructed automaton A_ϕ accepts the right floating language.

□

As the recursive event-clock automata subsume the formalisms that define the counter-free real-time ω-regular languages, we propose to call the languages identified by recursive event-clock automata as follows:

Definition 4.64 The sets of timed state sequences definable by the formalisms of recursive event-clock automata form the class of *real-time ω-regular languages.*

Note that the last theorem and theorem 4.11, allow us to derive the following corollary:

Corollary 4.65 ($\mathsf{EventClockTA} \subset \mathsf{REventClockTA}$) *The class of recursive event-clock automata is strictly more expressive that the class of propositional event-clock automata.* □

From the base case of the last proof, we can see that if p in not real-time constrained in ϕ then p does not appear in the subautomata of A_ϕ.

Lemma 4.66 (Not Real-Time Constrained Propositions) *Let ϕ be an* EventClockTL *formula and p a proposition of ϕ that does not appear in the scope of a real-time operator ($\triangleright, \triangleleft$) then we can construct an* MEventClockTA *A_ϕ such that (i)* $\mathsf{FloatLang}(A_\phi) = \mathsf{FloatLang}(\phi)$ *and (ii) p does not appear in the proposition used by subautomata of A_ϕ.*

We will use this property to determine how to introduce second-order quantification within real-time logics in the following section.

5 Adding Counting and Beyond

5.1 Introduction

In this section, we show how to close the gap between the counter-free real-time regular languages identified in section 3, and the (counter) real-time regular languages identified in the section 4. We will show that there are two ways to bridge this expressiveness gap.

The first way, is to add *automaton operators* to the real-time logics EventClockTL and MetricIntervalTL, giving respectively, E-EventClockTL and E-MetricIntervalTL. This is very similar to the situation in the temporal formalisms where it has been shown in [Wol83] that LTL can be extended with Büchi automata operators giving the logic E-TL which is able to express, in contrast with LTL, all regular languages. The only difference is that we need floating automata here because we must be able to look in the past. So E-EventClockTL and E-MetricIntervalTL define exactly the same class of real-time languages than the recursive event-clock automata.

The second way consists of adding *second-order quantification* to EventClockTL, MetricIntervalTL and $\mathsf{MinMaxML}_1$. But here the situation, surprisingly, is very different from the situation in untimed languages. In untimed languages, second-order quantification can be added without restriction and close the gap between counter-free and counter regular languages. In real-time, we will see that adding unrestricted second-order quantification leads to a fully undecidable formalism: neither satisfiability, nor validity are decidable, and the resulting formalisms are strictly more expressive than timed automata. But we will show that if we *slightly restrict* the use of second-order quantification, we obtain fully decidable formalisms called Q-EventClockTL, Q-MetricIntervalTL and $\mathsf{MinMaxML}_2$. Interestingly, those three formalisms define exactly the class of counter real-time regular languages as recursive event-clock automata.

We will show that the results that we have obtained are *sharp* in the sense that small relaxations of the syntactical restrictions that we have imposed to our formalisms either lead to formalisms that are as expressive as timed automata, or to fully undecidable formalisms. In particular, we will show that only adding *outermost second-order quantification*, called here *projection*, to all the fully decidable formalisms previously defined, leads to formalisms as expressive as timed automata and have thus a non decidable validity problem. As all those formalisms define the same class of real-time languages, we call this class "*projected real-time languages regular languages*". We also study two other relaxations that lead to fully undecidable formalisms.

5.2 Adding the Ability to Count

5.2.1 Adding Automata Operators

In this section, we give the definition of the syntax and semantics of the real-time logics EventClockTL and MetricIntervalTL extended with monitored floating automata operators (or equivalently add floating automata instead of their monitored version).

Definition 5.1 (E-EventClockTL-Syntax) The formulas of the extended event clock temporal logic E-EventClockTL are defined as for EventClockTL, see definition 3.16, with the following additional clause:

$$\phi ::= A(\phi_1, \ldots, \phi_n)$$

where $A = (Q, Q_0, Q_M, \delta, \Sigma, \lambda, Q_F)$ is a monitored floating automaton with $\Sigma = \{\phi_1, \phi_2, \ldots, \phi_n\}$ is the alphabet of A, $\lambda : Q \to \Sigma$ is the labeling function that labels each location of A with a E-EventClockTL formula, other elements are as for monitored floating automata, see definition 4.23. □

We define the semantics of the automata operators as follows:

Definition 5.2 (E-EventClockTL-Semantics) Let ϕ be an E-EventClockTL formula and let κ be a timed state sequence whose propositional symbols contain all propositions that occur in ϕ. The formula ϕ *holds* at time $t \in \mathbb{R}^+$ of κ, denoted $(\kappa, t) \models \phi$, according to the following definition:

For the operators of the logic EventClockTL, see definition 3.17;
$(\kappa, t) \models A(\phi_1, \phi_2, \ldots, \phi_n)$ iff there is an infinite t-monitored run ρ of A on κ that respects:

(1) *Covering, Start, Consecution, Monitoring and Acceptance* are as for monitored floating automata, see definition 4.24;
(2) *Constraint*: for all $t \in \mathbb{R}^+$, $(\kappa, t) \models \lambda(\rho(t))$;

□

Let us now turn to the extension of MetricIntervalTL.

Definition 5.3 (E-MetricIntervalTL-Syntax) The formulas of the extended metric interval temporal logic E-EventClockTL are defined as for MetricIntervalTL, see definition 3.9, with the following additional clause:

$$\phi ::= A(\phi_1, \ldots, \phi_n)$$

where $A = (Q, Q_0, Q_M, \delta, \Sigma, \lambda, Q_F)$ is a monitored floating automaton where $\Sigma = \{\phi_1, \phi_2, \ldots, \phi_n\}$ is the alphabet of A, $\lambda : Q \to \Sigma$ is the labeling function that labels each location of A with a E-MetricIntervalTL formula, other elements are as for monitored floating automata, see definition 4.23. □

We define the semantics of the automata operators as follows:

Definition 5.4 (E-MetricIntervalTL-semantics) Let ϕ be an E-MetricIntervalTL formula and let κ be a timed state sequence whose propositional symbols contain all propositions that occur in ϕ. The formula ϕ *holds* at time $t \in \mathbb{R}^+$ of κ, denoted $(\kappa, t) \models \phi$, according to the following definition:

For the operators of the logic MetricIntervalTL, see definition 3.11;
$(\kappa, t) \models A(\phi_1, \phi_2, \ldots, \phi_n)$ iff there is an infinite t-monitored run ρ of A on κ that respects:

(1) *Covering, Start, Consecution, Monitoring, Acceptance* are as for monitored floating automata, see definition 4.24;
(2) *Constraint*: for all $t \in \mathbb{R}^+$, $(\kappa, t) \models \lambda(\rho(t))$;

□

We will study the expressiveness and decidability results of those logics in the following sections.

5.2.2 Adding Second-Order Quantification

The *quantified temporal logics* Q-EventClockTL and Q-MetricIntervalTL are defined by adding second order quantification to EventClockTL and MetricIntervalTL in a restricted way.

Definition 5.5 (Q-EventClockTL-**Syntax**) The formulas of the quantified event clock temporal logic Q-EventClockTL are defined as for EventClockTL, see definition 3.16, with the following additional clause:

$$\phi ::= \exists p \cdot \psi$$

where p is a proposition which, inside the formula ψ, does not occur within the scope of a history or prophecy operator. □

We now define the semantics of the additional clause:

Definition 5.6 (Q-EventClockTL-**Semantics**) Let ϕ be an Q-EventClockTL formula and let κ be a timed state sequence whose propositional symbols contain all propositions that occur freely in ϕ. The formula ϕ *holds* at time $t \in \mathbb{R}^+$ of κ, denoted $(\kappa, t) \models \phi$, according to the following definition:

For the operators of the logic EventClockTL, see definition 3.17;
$(\kappa, t) \models \exists p \cdot \phi$ iff there is a $\{p\}$-extension of κ, noted κ^p, such that $(\kappa^p, t) \models \phi$;

□

Similarly, we define the second order quantification extension of MetricIntervalTL as follows:

Definition 5.7 (Q-MetricIntervalTL-**Syntax**) The formulas of the quantified metric interval temporal logic E-MetricIntervalTL are defined as for MetricIntervalTL, see definition 3.9, with the following additional clause:

$$\phi ::= \exists p \cdot \psi$$

where p is a proposition which, inside the formula ψ, does not occur within the scope of a real-time operator with interval different from $(0, \infty)$. □

We now define the semantics of the additional clause:

Definition 5.8 (Q-MetricIntervalTL-**Semantics**) Let ϕ be an Q-MetricIntervalTL formula and let κ be a timed state sequence whose propositional symbols contain all propositions that occur freely in ϕ. The formula ϕ *holds* at time $t \in \mathbb{R}^+$ of κ, denoted $(\kappa, t) \models \phi$, according to the following definition:

For the operators of the logic MetricIntervalTL, see definition 3.11;
$(\kappa, t) \models \exists p \cdot \phi$ iff there is a $\{p\}$-extension of κ, noted κ^p, such that $(\kappa^p, t) \models \phi$;

□

Similarly, we introduce second-order quantification in the real-time monadic theory that we have defined in section 3.4.

Definition 5.9 ($\mathsf{MinMaxML}_2$-**Syntax**) The formulas of the *Second-Order Real-Time Monadic Logic over the Reals* $\mathsf{MinMaxML}_2$ are defined as for $\mathsf{MinMaxML}_1$, definition 3.20, with the following additional clause:

$$\Phi ::= \exists p \cdot \Psi$$

where p is a monadic predicate which, inside the formula Ψ, does not occur within the scope of a real-time quantifier $\mathsf{Min}, \mathsf{Max}$. □

The semantics of the additional clause is as usual:

Definition 5.10 ($\mathsf{MinMaxML}_2$-**Semantics**) Let Φ be an $\mathsf{MinMaxML}_2$ formula and let κ be a timed state sequence whose propositional symbols contain all propositions that occur freely in Φ. The formula Φ *holds* in the pair (κ, α), denoted $(\kappa, \alpha) \models \Phi$, according to the following definition:

For the operators and terms of the logic $\mathsf{MinMaxML}_1$, see definition 3.21 and definition 3.22;
$(\kappa, \alpha) \models \exists p \cdot \Phi$ iff there is a $\{p\}$-extension of κ, noted κ^p, such that $(\kappa^p, \alpha) \models \Phi$;

□

5.2.3 Expressiveness Results

From the theorem 3.38 and the way we have defined $\mathsf{E\text{-}EventClockTL}$ and $\mathsf{E\text{-}MetricIntervalTL}$, we have the following corollary:

Corollary 5.11 ($\mathsf{E\text{-}EventClockTL} = \mathsf{E\text{-}MetricIntervalTL}$) *The logics* $\mathsf{E\text{-}EventClockTL}$ *and* $\mathsf{E\text{-}MetricIntervalTL}$ *are equivalently expressive.*

Similarly, we obtain the following corollary for $\mathsf{Q\text{-}EventClockTL}$, $\mathsf{Q\text{-}MetricIntervalTL}$ and $\mathsf{MinMaxML}_2$:

Corollary 5.12 ($\mathsf{Q\text{-}EventClockTL} = \mathsf{Q\text{-}MetricIntervalTL} = \mathsf{MinMaxML}_2$) *The logics* $\mathsf{Q\text{-}EventClockTL}$, $\mathsf{Q\text{-}MetricIntervalTL}$ *and* $\mathsf{MinMaxML}_2$ *are equivalently expressive.*

So, what will be proved for $\mathsf{Q\text{-}EventClockTL}$, can be derived for $\mathsf{Q\text{-}MetricIntervalTL}$ and for $\mathsf{MinMaxML}_2$.
Let us now study the relation that exists between our quantified logics and the formalisms of recursive event-clock automata:

Lemma 5.13 ($\mathsf{REventClockTA} \subseteq \mathsf{Q\text{-}EventClockTL}$) *For every* $\mathsf{REventClockTA}$ A*, we can construct a congruent* $\mathsf{Q\text{-}EventClockTL}$ *formula* ϕ_A*, that is for every* TSS κ*, every time* $t \in \mathbb{R}^+$*:* $\mathsf{Accept}_A(\kappa, t)$ *iff* $(\kappa, t) \models \phi_A$*.*

Proof. Using the equivalence result for $\mathsf{REventClockTA}$ and $\mathsf{MEventClockTA}$ given by theorem 4.28, we can show that for every $\mathsf{MEventClockTA}$ A, we can construct a congruent $\mathsf{Q\text{-}EventClockTL}$ formula ϕ_A. We reason by induction on the level of the $\mathsf{MEventClockTA}$ A.
Base case. The automaton $A = (Q, Q_0, Q_M, \delta, \mathcal{P}, \lambda, Q_F)$ is a monitored floating automata, i.e. $\mathsf{level}(A) = 0$. In that case, the formula ϕ_A is constructed from the following formulas:

- let $\mathsf{Controle}$ be the following propositional formula: $\underline{\bigvee}_{q\in Q} at_q$, where $\underline{\vee}$ denotes an exclusive or and the proposition at_q intuitively means that the control resides in location q. $\mathsf{Controle}$ means that at each time during a run, the control of the automaton resides in one and only one location.

- let Init be the following formula:

$$\neg \ominus \top \rightarrow \underline{\bigvee}_{q\in Q_0} at_q$$

  that expresses the initially ($\neg \ominus \top$) the control of the automaton must reside in an initial location;

- let $\mathsf{Transition}$ be the following formula:

$$\bigwedge_{q\in Q} at_q \rightarrow \begin{array}{l} 1.\wedge\ at_q \mathsf{W} \bigvee_{(q,q_2)\in\delta} at_{q_2} \\ 2.\wedge\ at_q \mathsf{Z} \bigvee_{(q_2,q)\in\delta} at_{q_2} \end{array}$$

  that expresses the transition relation.

- let $\mathsf{Monitoring}$ be the following formula:

$$\bigvee_{q\in Q_M} at_q$$

  that is true when the control of the automaton is in a monitored location;

- let $\mathsf{Labelling}$ be the following formula:

$$\bigwedge_{q\in q} at_q \rightarrow (\lambda(q))^T$$

  where $(\lambda(q))^T$ is as follows:

$$\bigwedge_{\psi\in\lambda(q)} \psi^T \wedge \bigwedge_{\psi\in\mathsf{Limit}(\mathcal{P})\setminus\lambda(q)} \neg\psi^T$$

and:

- $(p)^T = p$ for $p \in \mathcal{P}$;
- $(\overrightarrow{p})^T = \bigcirc p$;
- $(\overleftarrow{p})^T = \ominus p$;
- $(\top)^T = \top$;
- $(\overrightarrow{\top})^T = \bigcirc\top$;
- $(\overleftarrow{\top})^T = \ominus\top$.

- let $\mathsf{Acceptance}$ be the following formula: for the generalized Büchi acceptance $Q_F = \{F_1, \dots, F_n\}$: $\bigwedge_{F_i \in \mathcal{F}} \Box\Diamond \bigvee_{q \in F_i} at_q$

The formula ϕ_A that corresponds to the monitored floating automaton A is:

$$\begin{array}{ll} \exists at_{q_0}, \dots, at_{q_n} : & 1. \wedge \boxminus\mathsf{Control} \wedge \Box\mathsf{Control} \wedge \mathsf{Control} \\ & 2. \wedge \Diamond\mathsf{Init} \\ & 3. \wedge \boxminus\mathsf{Transition} \wedge \Box\mathsf{Transition} \wedge \mathsf{Transition} \\ & 4. \wedge \mathsf{Monitoring} \boxminus \mathsf{Labeling} \wedge \Box\mathsf{Labeling} \wedge \mathsf{Labeling} \\ & 5. \wedge \mathsf{Acceptance} \end{array}$$

Induction case. By induction hypothesis, for every sub-automaton $B \in \mathsf{SUB}(A)$, we are able to construct a congruent formula ϕ_B. Let us show that we can do it for A too. The only difference between an MEventClockTA and a monitored floating automata is the ability of MEventClockTA to use clock constraints in their labeling function. We define the function T that given a label χ of A, return the right EventClockTL formula. The label χ is a set of literals, more precisely, $\chi \subseteq \mathsf{Limit}(\mathcal{P}^A \cup \mathcal{A}^A)$. The construction is as for the base case, we only have to show how to deal with atomic clock constraints. We treat atomic real-time constraints as follows:

- $\psi = y_B \sim c$ then $\psi^T = \rhd_{\sim c}\phi_B$;
- $\psi = x_B \sim c$ then $\psi^T = \lhd_{\sim c}\phi_B$;

By examining the construction above, it is easy to see that the existentially quantified proposition, i.e. $at_{q_0}, \dots, at_{q_n}$ do not appear in the scope of a real-time operator, so the formula ϕ_A is in Q-EventClockTL. □

We now prove the reverse lemma:

Lemma 5.14 (Q-EventClockTL $\subseteq$ REventClockTA) *For every* Q-EventClockTL *formula ϕ, we can construct a congruent* REventClockTA *automaton A_ϕ, that is for every* TSS *κ, every time $t \in \mathbb{R}^+$:* $\mathsf{Accept}_{A_\phi}(\kappa, t)$ *iff* $(\kappa, t) \models \phi$.

Proof. By theorem 4.28, we can consider monitored recursive event clock automata in the proof. In the proof of lemma 4.63, we have shown that for every EventClockTL formula ϕ, we can construct a congruent MEventClockTA A_ϕ, from that proof, it can easily be shown that MEventClockTA are closed under all EventClockTL operators. Further, in lemma 4.52 it has been shown that MEventClockTA are partially closed under projection: a proposition that does not appear in a sub-automaton can be projected. So as quantified propositions do not appear, by definition, in the scope of real-time operators, they do not appear into a sub-automaton, see lemma 4.63, and thus can be projected. □

From the two previous lemmas and corollary 5.12, we obtain the following theorem:

Theorem 5.15 *The logics* Q-EventClockTL*,* Q-MetricIntervalTL *and* $\mathsf{MinMaxML}_2$ *have the same expressive power as* REventClockTA *automata.* □

And thus, as we have translation procedures between those formalisms, we have:

Theorem 5.16 *The satisfiability problems for* Q-EventClockTL*,* Q-MetricIntervalTL *and* $\mathsf{MinMaxML}_2$ *are decidable.* □

Since already the untimed quantified temporal logic Q-TL is non-elementary [Sis83], so are the satisfiability problems for Q-EventClockTL and Q-MetricIntervalTL.

Theorem 5.17 *The satisfiability problems for* Q-EventClockTL*,* Q-MetricIntervalTL *and* $\mathsf{MinMaxML}_2$ *are* NonElem. □

Let us now turn to the logics E-EventClockTL and E-MetricIntervalTL. Again, by theorem 3.37 and the definition of E-EventClockTL and E-MetricIntervalTL, we have the following corollary:

Corollary 5.18 (E-EventClockTL = E-MetricIntervalTL) *The logics* E-EventClockTL *and* E-MetricIntervalTL *are equivalently expressive.* □

So, what will be proved for E-EventClockTL, can be derived for E-MetricIntervalTL.

Lemma 5.19 (E-EventClockTL $\subseteq$ REventClockTA) *For every* E-EventClockTL *formula* ϕ*, we can construct a congruent* REventClockTA *automaton* A_ϕ*, that is for every* TSS κ*, every time* $t \in \mathbb{R}^+$*:* $\mathsf{Accept}_{A_\phi}(\kappa, t)$ *iff* $(\kappa, t) \models \phi$*.*

Proof. Again, thanks to the theorem 4.28 we can show that E-EventClockTL $\subseteq$ MEventClockTA. We already know that MEventClockTA are closed under all EventClockTL operators. With an adaptation of the techniques of [SVW85] (see also section 4.5) it can be shown that MEventClockTA are closed under monitored floating automata operators. □

The other direction is trivial:

Lemma 5.20 (REventClockTA $\subseteq$ E-EventClockTL) *For every* REventClockTA *automaton* A_ϕ*, we can construct a congruent* E-EventClockTL *formula* ϕ*, that for every* TSS κ*, every time* $t \in \mathbb{R}^+$*:* $(\kappa, t) \models \phi$ *iff* $\mathsf{Accept}_{A_\phi}(\kappa, t)$*.*

Thus the two formalisms are equally expressive.

Theorem 5.21 *The logic* E-EventClockTL *and automata* REventClockTA *are equally expressive.*

And thus,

Theorem 5.22 (All Equivalent) *The logics* E-EventClockTL*,* Q-EventClockTL*,* E-MetricIntervalTL*,* Q-MetricIntervalTL *and* $\mathsf{MinMaxML}_2$ *are all equivalent in expressive power to the formalisms of* REventClockTA*, and thus define the (counter) real-time regular languages.* □

As we have translation procedure between those formalisms, we have that:

Theorem 5.23 (E-EventClockTL **and** E-MetricIntervalTL**-Decidability)** *The logics* E-EventClockTL *and* E-MetricIntervalTL *are decidable.* □

Further, it can be shown that:

Theorem 5.24 (E-EventClockTL **and** E-MetricIntervalTL**-Complexity)** *The satisfiability problems for* E-EventClockTL *and* E-MetricIntervalTL$_{0,\infty}$ *are complete for* PSPACE. *The satisfiability problem for* E-MetricIntervalTL *is complete for* EXPSPACE. □

5.3 Projected Regular Real-Time Languages

In this section, we study the impact, in term of decidability and expressivity, of adding projection, i.e. outermost existential quantification, to the fully decidable that we have defined previously.

We will detail the introduction of projection into the logic of event clocks giving its projected version P-EventClockTL, the propositional (non recursive) event-clock automaton giving P-EventClockTA and the recursive event-clock automata giving P-REventClockTA. Using equivalence results that we have presented above, we derive implicitly all the corollaries for the other formalisms.

5.3.1 Projected Event-Clock Temporal Logic

We define the syntax and semantics of this logic as follows:

Definition 5.25 (P-EventClockTL**-Syntax)** The formulas of the *projected event clock temporal logic* P-EventClockTL are defined by the following clause:

$\exists p_1, \ldots, p_n \cdot \phi$

where ϕ is an EventClockTL-formula, see definition 3.16, and $p_1, \ldots, p_n$ are propositional symbols. □

Let us note that, in contrast with the definition of Q-EventClockTL, we allow in P-EventClockTL that quantified propositions appear in the scope of real-time operators. But on the other hand, quantification is only allowed as the outermost operator. The semantics of second-order existential quantification is the expected one:

Definition 5.26 (P-EventClockTL**-Semantics)** Let ϕ be an P-EventClockTL formula and let κ be a timed state sequence whose propositional symbols contain all propositions that occur freely in ϕ. The formula ϕ *holds* at time $t \in \mathbb{R}^+$ of κ, denoted $(\kappa, t) \models \phi$, according to the following definition:

For the operators of the logic EventClockTL, see definition 3.17;
$(\kappa, t) \models \exists p_1, \ldots, p_n \cdot \phi$ iff there is a $\{p_1, \ldots, p_n\}$-extension of κ, noted $\kappa^{\{p_1,\ldots,p_n\}}$, such that $(\kappa^{\{p_1,\ldots,p_n\}}, t) \models \phi$;

□

The anchored language of the P-EventClockTL formula $\exists p_1, \ldots, p_n \cdot \phi$ has the following relation with the anchored language of the EventClockTL formula ϕ:

Lemma 5.27 *If $\mathcal{P}$ is the set of propositions that appear in $\phi \in$* EventClockTL *and $\mathcal{P}' = \mathcal{P} \setminus \{p_1, \ldots, p_n\}$ then* AncLang$(\exists p_1, \ldots, p_n \cdot \phi) = \{\kappa \downarrow \mathcal{P}' \mid \kappa \in$ AncLang$(\phi)\}$. □

5.3.2 Projected (Propositional) Event-Clock Automata

The definitions for projected (propositional) event-clock automata are obtained in a similar way:

Definition 5.28 (P-EventClockTA**-Syntax)** A *projected (propositional) event-clock automaton* is a pair $(A, \{p_1 \ldots p_n\})$ that consists of a (propositional) event-clock automaton A and a set of propositions $\{p_1 \ldots p_n\}$. □

Definition 5.29 (P-EventClockTA**-Semantics)** The anchored language defined by a P-REventClockTA $(A, \{p_1 \ldots p_n\})$, with A defined on the set of propositions $\mathcal{P}$, is the $(\mathcal{P} \setminus \{p_1 \ldots p_n\})$-projections of TSS that belongs to the anchored language of A, that is, if we note $\mathcal{P}' = \mathcal{P} \setminus \{p_1, \ldots, p_n\}$, we have AncLang$((A, \{p_1 \ldots p_n\})) = \{\kappa \downarrow \mathcal{P}' \mid \kappa \in$ AncLang$(A)\}$. □

5.3.3 Projected Recursive Event-Clock Automaton

We now turn to the definition of projection into recursive event clock automata.

Definition 5.30 (P-REventClockTA-Syntax) A *projected recursive event-clock automaton* is a pair $(A, \{p_1 \dots p_n\})$ that consists of a recursive event-clock automaton A and a set of propositions $\{p_1 \dots p_n\}$. $\square$

Definition 5.31 (P-REventClockTA-Semantics) The anchored language defined by a P-REventClockTA $(A, \{p_1 \dots p_n\})$, with A defined on the set of propositions $\mathcal{P}$, is the $(\mathcal{P} \setminus \{p_1 \dots p_n\})$-projections of TSS that belongs to the anchored language of A, that is, if we note $\mathcal{P}' = \mathcal{P} \setminus \{p_1, \dots, p_n\}$, we have $\mathsf{AncLang}((A, \{p_1 \dots p_n\})) = \{\kappa \downarrow \mathcal{P}' \mid \kappa \in \mathsf{AncLang}(A)\}$. $\square$

5.3.4 Timed Automata

We briefly recall here the definition of timed automata. See [AD94] for a complete study of this formalism.

Definition 5.32 (Continuous Timed Automaton) A *continuous timed automaton* is a tuple $A = (Q, Q_0, C, E, \mathcal{P}, \lambda_p, \lambda_c, Q_F)$ where:

- Q is a finite set of locations;
- $Q_0 \subseteq Q$ is the subset of starting locations;
- C is a finite set of clocks;
- $E \subseteq Q \times 2^C \times Q$ a set of edges. An edge (q_1, ς, q_2) represents a transition from location q_1 to location q_2, ς is the subset of clocks that are reset when crossing the edge;
- $\mathcal{P}$ is a finite set of propositions;
- $\lambda_p : Q \to 2^{\mathcal{P}}$ is a labeling function which labels each location with the set of atomic propositions that are true in that location;
- $\lambda_c : Q \to \Delta(C)$ is a labeling function which assigns to each location a constraint of $\Delta(C)$ on the value of clocks that should be verified when staying in that location;
- Q_F is a set of accepting locations (Büchi acceptance condition).

$\square$

Definition 5.33 (TA-Timed Run) A *continuous timed run* of a continuous timed automaton A is an infinite sequence

$$\rho = (q_0, I_0) \to^{\varsigma_0} (q_1, I_1) \to^{\varsigma_1} \dots (q_n, I_n) \to^{\varsigma_n} \dots$$

- q_i are locations;
- $I_0 I_1 \cdots I_n \cdots$ is a sequence of intervals that partitions $\mathbb{R}^+$;
- $\varsigma_i \subseteq C$ are sets of clocks (to reset).

$\square$

Definition 5.34 (TA-Clock Value) The *value of a clock* $x \in C$ along a continuous timed run $\rho = (q_0, I_0) \to^{\varsigma_0} (q_1, I_1) \to^{\varsigma_1} \cdots$, at time $t \in I_i$, noted $\eta(\rho, t)(x)$, is defined as follows:

$$\eta(\rho, t)(x) = \begin{cases} t - r(I_j) & \text{if } x \in \varsigma_j \text{ and } \forall k \cdot j < k < i : x \notin \varsigma_k \\ t & \text{if } \forall j : 0 \leq j < i : x \notin \varsigma_j \end{cases}$$

We use $\eta(\rho, t)$ to denote the *clock valuation* at time t along ρ. $\square$

Definition 5.35 (Clock Constraints-Semantics) A clock constraint ψ is satisfied by a clock valuation η, noted $\eta \models \psi$, according to the following rules:

$\eta \models x \sim c$ iff $\eta(x) \sim c$, with $\sim \in \{<, \leq, =, \geq, >\}$;
$\eta \models \neg\psi$ iff $\eta \not\models \psi$;
$\eta \models \psi_1 \vee \psi_2$ iff $\eta \models \psi_1$ or $\eta \models \psi_2$.

□

Definition 5.36 (TA-Accepted Run) A continuous timed run $\rho = (q_0, I_0) \rightarrow^{\varsigma_0} (q_1, I_1) \rightarrow^{\varsigma_1} \ldots (q_n, I_n) \rightarrow^{\varsigma_n} \ldots$ is *accepted* by the continuous timed automaton $A = (Q, Q_0, C, E, \mathcal{P}, \lambda_p, \lambda_c, Q_F)$ when reading the TSS $\kappa = (\overline{s}, \overline{I})$ iff ρ respects the following requirements:

- *Starting.* The first location in ρ is a starting location of A, that is $q_0 \in Q_0$;
- *Consecution.* The continuous timed run ρ respects the transition relation of A, i.e. for all positions $i \geq 0$, we have that either $(q_i, \varsigma_i, q_{i+1}) \in E$, or $q_i = q_{i+1}$ and $\varsigma_i = \emptyset$ (stuttering steps are allowed);
- *Timing constraints.* The timing constraints about clocks are respected along ρ, that is, for every position $i \geq 0$, for all time $t \in I_i : \eta(\rho, t) \models \lambda_c(q_i)$;
- *Adequation.* The labels along the continuous timed run ρ are in adequation with the truth value of the propositions along the TSS κ, that is for all time $t \in \mathbb{R}^+$, $(\kappa, t) \models \lambda_p(\rho(t))$;

Further, we say that ρ is *accepting* if there exists infinitely many positions $i \geq 0$ such that $q_i \in Q_F$. We note $\mathsf{Accept}_A(\kappa)$ the fact that A has an accepted continuous timed run on κ. □

Definition 5.37 The *anchored language* of a continuous TA A is the set of TSS κ on which A has an accepted run, i.e. $\mathsf{AncLang}(A) = \{\kappa \in \mathsf{TSS}(2^{\mathcal{P}^A}) \mid \mathsf{Accept}_A(\kappa)\}$. □

Timed automata are closed under positive boolean operation but not under negation.

Theorem 5.38 (Closure under Union and Intersection) *[AD94] Timed automata are closed under union and intersection.* □

Theorem 5.39 (Non-Closure under Complement) *The formalism of timed automata is not closed under complement.* □

The emptiness problem of timed automata is decidable, on the other hand, its universality problem is undecidable.

Theorem 5.40 (TA-Emptiness) *[AD94] The emptiness problem for timed automata is* PSPACE-COMPLETE. □

But the universality problem, that is given a timed automaton, determine if it accepts all possible timed traces, is undecidable.

Theorem 5.41 (TA-Universality) *[AD94] The problem of universality for timed automaton is undecidable.* □

5.3.5 Expressiveness Results

In this paragraph, we show that adding projection to the fully decidable formalism is powerful. In fact, we will show that even if added to propositional event-clock automata we obtain a formalism which is expressively equivalent to timed automata. The same occurs with all the other formalisms that we have defined.

We now prove that adding projection to the logic $\mathsf{EventClockTL}$ extends is expressive power in such a way that $\mathsf{P\text{-}EventClockTL}$ at least as expressive as TA:

Lemma 5.42 ($\mathsf{TA} \subseteq \mathsf{P\text{-}EventClockTL}$) *For every continuous timed automaton A, we can compute a projected event clock temporal formula ϕ_A that defines exactly the anchored timed language defined by A, that is,* $\mathsf{AncLang}(\phi) = \mathsf{AncLang}(A)$

Proof. Let $A = (Q, Q_0, C, E, \mathcal{P}, \lambda_p, \lambda_c, Q_F)$ be the continuous timed automaton for which we want to construct the $\mathsf{P\text{-}EventClockTL}$ formula ϕ_A. We construct ϕ_A as follows:

- For each location $q \in Q$ we introduce the proposition at_q to express that the control of automaton resides in location q. During a run the control of an timed automaton A resides in one and only one location at a given time. This is expressed by the following formula:

$$F_Q \equiv \Box \underline{\bigvee}_{q \in Q} at_q$$

  with $Q = \{q_1, q_2, \ldots, q_n\}$

- The initial condition is expressed by the following formula:

$$F_{Q_0} \equiv \bigvee_{q \in Q_0} at_q$$

- the propositional labeling function λ_p is translated as follows:

$$F_{\mathcal{P}} \equiv \Box \bigwedge_{q \in q} (at_q \rightarrow \bigwedge_{p \in \lambda_p(q)} p \wedge \bigwedge_{p \in \mathcal{P} \setminus \lambda_p(q)} \neg p)$$

- The resetting of clocks can be expressed with the help of existentially quantified variables. For each clock $c_i \in C$, we associate a proposition that we note r_{c_i}. This proposition r_{c_i} will be true when and only when the clock c_i is reset. By definition of timed automata, clocks are reset when crossing edges, and implicitly at the initial moment. For each edge (q_i, λ_r, q_j) of the automaton, we introduce the proposition $cross_{(q_i, \lambda_r, q_j)}$ that is true iff the automaton crosses the edge between location q_i and location q_j.

$$F_{R_1} \equiv \Box \bigwedge_{(q_i, \lambda_r, q_j) \in E} cross_{(q_i, \lambda_r, q_j)} \leftrightarrow (at_{q_i} \wedge \bigcirc al_{q_j}) \vee (at_{q_j} \wedge \ominus al_{q_i})$$

$$F_{R_2} \equiv \Box [\bigwedge_{(q_i, \lambda_r, q_j) \in E} (cross_{(q_i, \lambda_r, q_j)} \rightarrow \bigwedge_{c \in \lambda_r} r_c)] \wedge \neg \ominus \top \rightarrow \bigwedge_{c \in C} r_c$$

  when the edge (q_i, λ_r, q_j) is crossed, the clocks that decorates the edge are reset (clocks are only reset when crossing edges that are labeled by the clock and initially).

$$F_{R_3} \equiv \Box [\bigwedge_{c \in C} r_c \rightarrow \bigvee_{(q_i, \gamma, q_j) \in R(c)} cross_{(q_i, \lambda_r, q_j)} \vee \neg \ominus \top]$$

  where $R(c)$ is the set of edges where the clock c is reset, i.e. $R(c) = \{(q_i, \lambda_r, q_j) \mid (q_i, \lambda_r, q_j) \in E \wedge c \in \lambda_r\}$.

$$F_R \equiv F_{R_1} \wedge F_{R_2} \wedge F_{R_3}$$

- The consecution rule is expressed by the following formula:

$$F_E \equiv \Box \bigwedge_{q \in Q} (at_q \rightarrow at_q \mathsf{W} \bigvee_{q' \in S_q} at_{q'})$$

where S_q is the set of locations that are successors of q in A, i.e. $S_q = \{q' | (q, q') \in E\}$;

- The semantics of the time constraint labeling function λ_c is translated as follows:

$$F_C \equiv \Box \bigwedge_{q \in Q} (at_q \rightarrow T(\lambda_C(q)))$$

where T is defined as:

  - $T(\psi_1 \vee \psi_2) = T(\psi_1) \vee T(\psi_2)$
  - $T(\neg\psi) = \neg T(\psi)$
  - $T(x \sim c) = \lhd_{\sim c} r_x$

- The acceptance condition constraint is defined as follows:

$$F_{Q_F} \equiv \bigwedge_{F_i \in Q_F} \Box\Diamond \bigvee_{q \in F_i} at_q$$

The P-EventClockTL formula whose anchored language is exactly the timed state sequences accepted by A is:

$$\exists A, C, R(F_L \wedge F_{L_0} \wedge F_E \wedge F_{\lambda_{\mathcal{P}}} \wedge F_R \wedge F_{\lambda_C} \wedge F_{\mathsf{F}})$$

where :

- $A = \{at_q | q \in Q\}$;
- $C = \{cross_{(q_i, \lambda_r, q_j)} | (q_i, \lambda_r, q_j) \in E\}$;
- $R = \{r_c | c \in C\}$.

$\Box$

We now take a look at the other direction:

Lemma 5.43 (P-EventClockTL $\subseteq$ TA) *For every projected event clock temporal formula $\psi \equiv \exists p_1, \ldots, p_n \cdot \phi$, we can compute a timed automaton A_ψ that defines exactly the anchored language defined by ϕ, that is,* $\mathsf{AncLang}(A_\psi) = \mathsf{AncLang}(\psi)$

Proof. By theorem 3.33, we know that for every EventClockTL formula, we can compute an equivalent MetricIntervalTL formula ϕ^T. By theorem **??**, we know that for this formula ϕ^T, we can construct an equivalent timed automaton A_{ϕ^T} which is also equivalent to ϕ. Finally, as timed automata are closed under projection, it follows that we can construct a timed automata for the P-EventClockTL formula $\exists p_1, \ldots, p_n \cdot \phi$ simply by projecting $p_1, \ldots, p_n$ in A_{ϕ^T}. $\Box$

From the two previous lemmas, we derive the following theorem:

Theorem 5.44 (P-EventClockTL = TA) *The formalisms of projected event clock temporal logic and timed automata are equally expressive to define anchored languages.*

Let us now turn to the characterization of the expressive power of the projected (propositional) event-clock automata. First, we have the following lemma:

Lemma 5.45 (P-EventClockTL $\subseteq$ P-EventClockTA) *For every projected event clock temporal formula $\exists p_1, \ldots, p_n \cdot \phi$, we can compute a projected propositional event-clock automaton (A_ϕ, Q) that defines exactly the anchored language defined by ϕ, that is,* $\mathsf{AncLang}((A_\phi, Q)) = \mathsf{AncLang}(\phi)$.

Proof. In [RS97], it is shown that for every formula $\phi \in$ EventClockTL, it is possible to construct an propositional event-clock automaton A_ϕ that accepts exactly the Hintikka sequences of ϕ. Remember that Hintikka sequences are just H-extensions of TSS that belongs to the anchored language of ϕ and the $\mathcal{P}$-projections of those TSS, where $\mathcal{P}$ is the set of propositions appearing in ϕ are exactly the TSS that belongs to AncLang(ϕ). So the following P-EventClockTA $(A_\phi, H \cup \{p_1, \dots, p_n\})$ is exactly the projected automaton we are looking for. □

We now show that the formalism of projected propositional event clock automata defines anchored languages that can be defined using timed automata:

Lemma 5.46 (P-EventClockTA $\subseteq$ TA) *For every projected propositional event-clock automaton $(A, \{p_1, \dots, p_n\})$, we can compute a timed automaton B that defines exactly the same anchored language, that is,* $\mathsf{AncLang}((A, \{p_1, \dots, p_n\})) = \mathsf{AncLang}(B)$.

Proof. In [AFH94], it is proved that for every propositional even-clock automaton we can construct a timed automaton that defines exactly the same anchored language. So for A, we can construct an equivalent timed automaton C. By lemma **??**, we know that we can construct B form C by projecting the set of propositions $\{p_1, \dots, p_n\}$. □

So we have the following corollary:

Corollary 5.47 *The formalism of* TA*,* P-EventClockTL*,* P-EventClockTA *are equally expressive to define anchored languages.*

Finally, we turn to the expressiveness of projected recursive event-clock automata:

Lemma 5.48 (P-REventClockTA $\subseteq$ P-EventClockTA) *For every projected recursive event-clock automaton $(A, \{p_1, \dots, p_n\})$, we can compute a projected propositional event clock automaton (B, Q) that defines exactly the same anchored language, that is,* $\mathsf{AncLang}((A, \{p_1, \dots, p_n\})) = \mathsf{AncLang}((B, Q))$.

Proof. First lemma 4.55 says that given an recursive event clock automaton A, we can construct a propositional event-clock automaton C that accepts exactly the timed Hintikka sequences of A. Let us note $\mathcal{P}'$ the set of propositions used by B, we now that $\{\kappa \downarrow \mathcal{P} \mid \kappa \in \mathsf{AncLang}(B)\} = \mathsf{AncLang}(A)$, so the following projected propositional event-clock automaton $(B, (\mathcal{P}' \setminus \mathcal{P}) \cup \{p_1, \dots, p_n\})$ accepts the desired anchored language. □

So, from the previous lemmas, we obtain the following lemma:

Theorem 5.49 *All the formalisms* TA*,* P-EventClockTA*,* P-REventClockTA *and* P-EventClockTL *define the same class of real-time languages.*

As all those formalisms define the same class of languages, we give it a name:

Definition 5.50 The class of real-time languages defined by TA, P-EventClockTA, P-REventClockTA and P-EventClockTL are called the *projected real-time regular languages.*

The proof that the projected formalisms are all equivalent to timed automata contains an effective translation, giving their decidability:

Theorem 5.51 (Projection and Decidability) *The projected formalisms* P-EventClockTA*,* P-REventClockTA *and* P-EventClockTL *have decidable decidable satisfiability (emptiness) problems and undecidable validity (universality) problems.*

Proof. The decidability of satisfiability follows directly, for each projected formalisms, from the fact that existential quantification does not change satisfiability. The undecidability of validity follows from the undecidability of the universality problem for timed automata, see theorem 5.41, and the equivalence of expressive power of the projected formalisms with timed automata, see theorem 5.49. □

5.4 Undecidable Extensions

In this section, we show that the result about decidability and expressiveness that we have obtained in the previous sections are *sharp* in the sense that if we liberalize the definitions of the previous formalisms we encounter undecidability problems.

First, in our second-order formalisms $\mathsf{MinMaxML}_2$, respectively in Q-EventClockTL, we have prohibited quantified monadic predicates, respectively propositions, from occurring within the scope of Min or Max quantifiers, respectively history or prophecy operators. We defined the unrestricted $\mathsf{MinMaxML}_2$ and Q-EventClockTL as follows:

Definition 5.52 (Unrestricted-Q-EventClockTL and $\mathsf{MinMaxML}_2$) The *unrestricted*-Q-EventClockTL logic is obtained by adding (unrestricted) second-order quantification to EventClockTL and the *unrestricted*-$\mathsf{MinMaxML}_2$ logic is obtained by adding (unrestricted) second-order quantification to $\mathsf{MinMaxML}_1$.

Obviously, we have the following lemma:

Lemma 5.53 *The logic unrestricted-*Q-EventClockTL *contains* P-EventClockTL *and is closed under boolean operations. The logic unrestricted-*$\mathsf{MinMaxML}_2$ *contains* P-$\mathsf{MinMaxML}_2$ *and is closed under boolean operations.*

The restriction on the use of second-order quantification is necessary for decidability. If, as seen above, we admit only *outermost existential* quantification (projection) over monadic predicates (propositions) that occur within the scope of real-time operators, we obtain a positively decidable formalism (satisfiability is decidable, but validity is not) which is expressively equivalent to timed automata. Consequently, if we admit full quantification over monadic predicates (propositions) that occur within the scope of real-time operators, then both satisfiability and validity are undecidable, and the formalism is expressively equivalent to boolean combinations of timed automata.

Theorem 5.54 *Formalisms that are able to express boolean combinations of projected formalisms have undecidable satisfiability and validity problems.*

So, as unrestricted $\mathsf{MinMaxML}_2$ and Q-EventClockTL allow the expression of boolean combinations of projected timed regular languages, we have the following theorem:

Theorem 5.55 *The logics unrestricted-*Q-EventClockTL *and unrestricted-*$\mathsf{MinMaxML}_2$ *have undecidable satisfiability and validity problems.*

We now turn to the restriction that we impose on $\mathsf{MinMaxML}_1$ formulas. A fully undecidable extension of $\mathsf{MinMaxML}_1$ is obtained by relaxing the restriction that in every formula of the form $(\mathsf{Min}\ t_1)(t_1 > t_2 \wedge \Psi(t_1)) \sim (t_2 + c)$ or $(\mathsf{Max}\ t_1)(t_1 < t_2 \wedge \Psi(t_1)) \sim (t_2 - c)$, the sub-formula $\Psi(t_1)$ contains no free occurrences of first-order variables other than t_1. If we suppress this restriction, it can be shown that the real-time temporal logic MetricTL can be embedded in $\mathsf{MinMaxML}_1$.

Definition 5.56 (Unrestricted-$\mathsf{MinMaxML}_1$) The formulas of *unrestricted*-$\mathsf{MinMaxML}_1$ are obtained from relaxing the constraints on the free variables occuring in the scope of Min − Max quantifiers.

For this unrestricted version of $\mathsf{MinMaxML}_1$, we have the following lemma:

Lemma 5.57 *For every formula of* MetricTL *there exists a congruent formula of* $\mathsf{MinMaxML}_1$.

Proof. We simply show that we are able to express the $\Diamond_{=c}$ operator of $\mathsf{MetricTL}$ (which is sufficient to obtain undecidability), other constructs of the logic are easier. The formula $\Diamond_{=1}p$ of $\mathsf{MetricTL}$ is expressed as follows in unrestricted-$\mathsf{MinMaxML}_1$:

$$\exists t_2 \cdot [\mathsf{Min}_{t_1} \cdot (t_1 > t \wedge t_1 = t_2) = t + 1] \wedge p(t_2)$$

In fact, $\exists t_2 \cdot \mathsf{Min}_{t_1} \cdot (t_1 > t \wedge t_1 = t_2) = t + 1$ forces t_2 to be equal to $t + 1$. □

Since $\mathsf{MetricTL}$ is undecidable [AH93], so are the satisfiability and validity problems for unrestricted $\mathsf{MinMaxML}_1$.

Theorem 5.58 (Undecidability) *The satisfiability and validity problems for unrestricted-*$\mathsf{MinMaxML}_1$ *are undecidable.* □

6 Conclusion

We have shown that $\mathsf{EventClockTL}$, when evaluated in timed state sequences, has exactly the same expressive power as $\mathsf{MetricIntervalTL}$. This nice result is surprising because $\mathsf{EventClockTL}$ and $\mathsf{MetricIntervalTL}$ are rather different logics, that propose orthogonal restrictions to reach decidability: $\mathsf{EventClockTL}$ allows punctuality constraints but restricts real-time constraints to refer to the next (last) time a formula will be (was) true, whereas $\mathsf{MetricIntervalTL}$ allows formulas to refer to any time where a formula will be true, but disallows punctuality constraints. In the process of proving the equivalence between the expressive powers of $\mathsf{EventClockTL}$ and $\mathsf{MetricIntervalTL}$, we have also shown that the PSPACE fragment of $\mathsf{MetricIntervalTL}$, that is $\mathsf{MetricIntervalTL}_{0,\infty}$, is expressively complete. Those results have been reinforced by the definition of a real-time first-order monadic theory, called $\mathsf{MinMaxML}_1$, that identifies exactly the same class of real-time languages as $\mathsf{MetricIntervalTL}$ and $\mathsf{EventClockTL}$. As two very different logics and a classical theory identify the same class of fully decidable real-time languages, we have proposed to call this class of languages the "counter-free real-time regular languages".

We have also shown that the expressive powers of $\mathsf{EventClockTL}$ and the propositional event-clock automata, as proposed in [AFH94], are incomparable. To remedy this situation, we have proposed to generalize the concept of event clock by allowing, recursively, automata as events. More precisely, these automata reset a clock when they enter their monitored locations. This yields a formalism that we have called the *recursive* event-clock automata, noted $\mathsf{REventClockTA}$. These automata subsume the expressive power of the logic $\mathsf{EventClockTL}$, and keep all the nice properties of the propositional version, namely: closure under *all* boolean operations and decidability of *both* the emptiness and universality problems. Further, we have shown that by adding the ability to count to the formalisms that identify the "counter-free real-time regular languages", we obtain formalisms that recognize the same class of languages than our $\mathsf{REventClockTA}$. So, we proposed to call this class of languages the "real-time regular languages". The introduction of second-order quantification into real-time logics requires some care: second-order quantification can be used *outside* or *inside* real-time operators but not *through* real-time operators. This is quite different from the qualitative case, where no restriction on second-order quantification is needed. We have shown that this result is *sharp* in the sense that: *first*, it is exactly what we need to bridge the gap between counter-free and counting real-time regular languages, *second*, even small relaxations of this restriction lead to lose full decidability and closure under negation. Finally, we have shown that adding projection, that is an outermost second-order quantification, to counter-free or (counting) real-time regular languages, leads to formalisms expressively equivalent to timed automata. Therefore, we proposed to call these languages, the "projected real-time regular languages". This class is not closed under negation and the corresponding formalisms are only positively decidable. All those results are summarized in the following tables:

	Languages	Temporal logics	Monadic theories	Finite automata
Fully decidable				
1	$\mathbb{R}$-*timed counter-free* ω-*regular*	MetricIntervalTL = EventClockTL	$\mathsf{MinMaxML}_1$	
2	$\mathbb{R}$-*timed* ω-*regular*	Q-MetricIntervalTL = Q-EventClockTL = E-MetricIntervalTL = E-EventClockTL	$\mathsf{MinMaxML}_2$	REventClockTA

(projection, or outermost existential quantification, is indicated by P-):

Positively decidable				
3	*projection-closed* $\mathbb{R}$-*timed* ω-*regular*	P-EventClockTL	$\mathsf{P\text{-}MinMaxML}_2$ $= \mathcal{L}d^{\leftrightarrow}$	P-REventClockTA = TA

References

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994. Preliminary version appears in Proc. 17th ICALP, 1990, LNCS 443.

[AFH91] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proceedings of the Tenth Annual Symposium on Principles of Distributed Computing*, pages 139–152. ACM Press, 1991.

[AFH94] R. Alur, L. Fix, and T.A. Henzinger. A determinizable class of timed automata. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 1–13. Springer-Verlag, 1994.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[AH90] R. Alur and T.A. Henzinger. Real-time logics: complexity and expressiveness. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 390–401. IEEE Computer Society Press, 1990.

[AH92] R. Alur and T.A. Henzinger. Back to the future: towards a theory of timed regular languages. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 177–186. IEEE Computer Society Press, 1992.

[AH93] R. Alur and T.A. Henzinger. Real-time logics: complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993. Special issue for LICS 90.

[AH94] R. Alur and T.A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Proceedings of the 13th Annual Symposium on Principles of Programming Languages*, pages 173–183. ACM Press, 1986.

[Büc62] J.R. Büchi. On a decision method in restricted second-order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Proceedings of the First International Congress on Logic, Methodology, and Philosophy of Science 1960*, pages 1–11. Stanford University Press, 1962.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the Seventh Annual Symposium on Principles of Programming Languages*, pages 163–173. ACM Press, 1980.

[Kam68] J.A.W. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California at Los Angeles, 1968.

[RS97] J.-F. Raskin and P.-Y. Schobbens. State clock logic: a decidable real-time logic. In O. Maler, editor, *HART 97: Hybrid and Real-time Systems*, Lecture Notes in Computer Science 1201, pages 33–47. Springer-Verlag, 1997.

[Sis83] A.P. Sistla. *Theoretical Issues in the Design and Verification of Distributed Systems.* PhD thesis, Harvard University, 1983.

[SVW85] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *Proc. 10th Int. Colloquium on Automata, Languages and Programming*, volume 194, pages 465–474, Nafplion, July 1985. Lecture Notes in Computer Science, Springer-Verlag.

[Wil94] T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *FTRTFT 94: Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 863, pages 694–715. Springer-Verlag, 1994.

[Wol82] P. Wolper. *Synthesis of Communicating Processes from Temporal-Logic Specifications.* PhD thesis, Stanford University, 1982.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.

[WVS83] P. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 185–194. IEEE Computer Society Press, 1983.