Timed-Automata-Based Verification of MITL over Signals

Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, Benjamin Monmege

Timed-automata-based verification of MITL over signals

Thomas Brihaye[1], Gilles Geeraerts[2], Hsi-Ming Ho[1], and Benjamin Monmege[3]

1 **Université de Mons, Belgium,** {thomas.brihaye,hsi-ming.ho}@umons.ac.be
2 **Université libre de Bruxelles, Belgium,** gigeerae@ulb.ac.be
3 **Aix Marseille Univ, CNRS, LIF, France,** benjamin.monmege@univ-amu.fr

Abstract

It has been argued that the most suitable semantic model for real-time formalisms is the non-negative real line (signals), i.e. the continuous semantics, which naturally captures the continuous evolution of system states. Existing tools like Uppaal are, however, based on ω-sequences with timestamps (timed words), i.e. the pointwise semantics. Furthermore, the support for logic formalisms is very limited in these tools. In this article, we amend these issues by a compositional translation from Metric Temporal Interval Logic (MITL) to signal automata. Combined with an emptiness-preserving encoding of signal automata into timed automata, we obtain a practical automata-based approach to MITL model-checking over signals. We implement the translation in our tool MightyL and report on case studies using LTSmin as the back-end.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.1.1 Models of Computation

Keywords and phrases real-time temporal logic, timed automata, real-time systems

Digital Object Identifier 10.4230/LIPIcs.TIME.2017.?

1 Introduction

Many computer programs nowadays control critical applications, and need to enforce complex requirements in order to guarantee safe, dependable and efficient operation of the whole system. Among these requirements, real-time specifications (such as 'every request is eventually followed by an acknowledgement *within* 3 time units') are common. In this framework, computer interact with an environment that is intrinsically continuous, and ensuring thin real-time constraints is known to be a very difficult task.

Different kinds of formalisms have been proposed over the past 30 years to *specify* those real-time models (often by means of automata) and requirements (usually by means of some logic language). On the automata side, the model of timed automata [2] is arguably widely accepted today, a success which is due in part to the tool support provided by Uppaal [35] and other verification tools such as Kronos [12], TiAMo [11], ... As far as logics are concerned, several proposals have been made in the literature during the past 30 years (such as MTL [33], TPTL [7], TCTL [1], ...) but the recent research seems to focus mainly on MTL, for theoretical reasons (we think here of the works of Ouaknine and Worrell on the decidability of MTL [38]); and on MITL [4] for more practical motivations [10,13,14,16,31,36].

Indeed, since its introduction in 1996, MITL has been advocated as a good '*trade-off between realistic modelling of time and feasible verification of timing properties*' [4]. MITL is at the same time a real-time extension of LTL, the most widely accepted logic in the non-real-time case; and a restriction of MTL, whose expressive power makes it undecidable in most practical cases [6,38]. Unfortunately, tool support for MITL is still lacking today,

albeit MITL's clear practical interest (and indeed, the need for such tool support is repeatedly emphasised in several papers [4, 10, 36]). UPPAAL, the most prominent real-time model checker, supports only a restricted subset of TCTL; and the alternatives are either not publicly available, or too restricted, or too experimental (see the *related work* hereinafter for a more comprehensive picture). We believe this is due to the relative lack of maturity of automata-based support for MITL, at least when compared with LTL.

Another point of debate in the community is the choice of the semantics for real-time models. The two different options are known as the *pointwise* and *continuous* semantics. In the pointwise semantics, executions of the system are timed words, i.e. sequences of pairs (timestamp, system state). That is, the system's states can only be observed at selected timestamps (which are non-negative real values). In the continuous semantics, executions are *signals*, i.e. sequences of contiguous intervals during which the states of the system does not change and can be continuously observed. While the pointwise semantics is the most common today (probably due to the success of timed automata which have initially been defined in this framework), it has been argued [8, 29] that the continuous semantics models time more faithfully, and it is indeed adopted in many works about control of hybrid systems [41], synthetic biology [9], etc. Apart from these practical considerations, the difference between these two semantics matters as it changes the expressive power of the logic[1]. For example, the following formula (asking that p holds exactly in an interval of the form $[0, a]$ for some $a \geq 0$) is satisfiable in the continuous semantics only: $p \wedge \mathbf{F}(\neg p) \wedge \mathbf{G}\big(\neg p \Rightarrow \mathbf{G}(\neg p)\big) \wedge \neg\big(p\,\mathcal{U}\,(\neg p)\big)$.

Contribution. In order to remedy the lack of comprehensive tool support for MITL in the *pointwise* semantics, we have recently introduced MIGHTYL [15], an efficient tool that turns MITL formulae into a network of timed automata (expressed in the UPPAAL language) accepting the same language. These timed automata can then be used to perform satisfiabilty or model-checking, using off-the-shelf model checkers such as UPPAAL or LTSMIN. The central point of the efficiency of our construction is its *compositional* feature: we output a network of timed automata (one per subformula) instead of a single, monolithic, one. In the present work, we extend this line of research to the realm of *continuous semantics* by revisiting the compositional translation of MITL into *signal automata* (i.e. automata akin to timed automata, but that accept signals instead of timed words).

More precisely, we introduce, in Section 3, a compositional translation that turns an MITL formula φ into a network of signal automata $\mathcal{C}_{init} \times \prod_\chi \mathcal{C}_\chi$, one for each subformula χ in φ, plus an extra signal automaton $\mathcal{C}_{init}$ (extending the ideas of our previous work [15] to the continuous setting). However, as is, this translation would not allow us to rely on the currently available tools for timed systems since most of them (and in particular, UPPAAL) rely on the pointwise semantics. So, in Section 4, we present an emptiness-preserving and compositional transformation from signal automata to timed automata (see Theorem 11). Concretely, given a signal automaton $\mathcal{A}$ modelling a system, and a property φ to be checked on $\mathcal{A}$, we can perform model-checking by: (i) building the network of signal automata $\mathcal{C}_{init} \times \prod_\chi \mathcal{C}_\chi$ from $\neg\varphi$ using the procedure of Section 3; (ii) translating, using the techniques of Section 4, $\mathcal{A}$, $\mathcal{C}_{init}$ and all $\mathcal{C}_\chi$ into corresponding *timed* automata $\mathcal{B}^{\mathcal{A}}$, $\mathcal{B}_{init}$ and $\mathcal{B}_\chi$ (for all subformulae χ) respectively; and (iii) checking (using a model-checker for timed automata) whether the language $\mathcal{B}^{\mathcal{A}} \times \mathcal{B}_{init} \times \prod_\chi \mathcal{B}_\chi$ is empty. If this is the case, then the properties of our translation ensure that this emptiness holds if and only if the language of $\mathcal{A} \times \mathcal{C}_{init} \times \prod_\chi \mathcal{C}_\chi$ is empty, which holds if and only if $\mathcal{A} \models \varphi$, by construction. We have implemented this approach as

[1] As for MTL, for instance, which becomes decidable on finite words in the pointwise semantics [38].

an extension of MightyL and report on experiments in Section 5. The preliminary results are very encouraging, as our approach compares well or outperforms previous approaches from the literature.

Related work. The most similar work to ours is [32] where the authors propose a compositional translation from MITL with past operators [5] to signal automata. The translation works by rewriting the input formula into one with only past operators using projections [22]. Each past subformula can then be handled by a simple component, and the resulting automaton is obtained by synchronising the components via newly introduced propositions. An advantage of this approach is that it directly supports past operators. Unfortunately, the rewriting step does not work for unbounded future operators; this severely limits the applicability of the translation (for example, the liveness property $\mathbf{GF}p$ cannot be expressed in the bounded-future fragment). Also, as far as we know, it has never been implemented. By contrast, while our translation deals only with future MITL, one may use projections to remove past operators from the input formula.

Compositional translations that support unbounded future operators also exist in the literature [21, 36, 37]. One difference of these with our translation is that they are formulated in terms of non-standard models such as *timed signal transducers* or *hierarchical timed automata*. This deviation from the more common models, we believe, has contributed to the lack of implementation of these translations.[2] Another difference is that the components constructed by these approaches are *testers* whereas those constructed by ours are *positive testers* [17, 40]; that is, suppose we introduce a new proposition p_χ for the subformula $\chi = \varphi_1 \,\mathcal{U}\, \varphi_2$, a tester enforces $p_\chi \Leftrightarrow \varphi_1 \,\mathcal{U}\, \varphi_2$ to hold at all times while a positive tester only enforces the weaker formula $p_\chi \Rightarrow \varphi_1 \,\mathcal{U}\, \varphi_2$ to hold at all times. This may affect the performance of verification algorithms [43]. Moreover, the weaker condition allows us to impose some minimality criteria on transitions for further performance gains (see Section 5).

The original translation from MITL to signal automata in [4] is a monolithic tableau-based procedure which follows roughly the same lines as the tableau-based translation from LTL to Büchi automata [27]: the locations of the resulting automaton are labelled by sets of subformulae, and the transitions between them are obtained by 'expanding' the labels. Like our translation, it also enforces minimality when generating transitions. However, the procedure is much more involved than the LTL counterpart and seems difficult to realise in practice. A simplified tableau-based translation is given in [25, 26] where an implementation—the only implementation of an MITL to signal automata translation we are aware of—is also reported. Nevertheless, the translation only works for the upper-bound fragment of MITL, and the tool is not publicly available.

Besides automata-based approaches, there are also proposals to apply SMT (Satisfiability Modulo Theories) solvers [19] to satisfiability/model-checking for MITL over signals [10, 31]. The SMT approach is straightforward to implement and there are publicly available tools. However, it is essentially a 'bounded model-checking' approach and therefore is inherently incomplete, unless very large (impractical) bounds are used.

2 Model-checking signal automata against MITL

This section introduces the main objects we study—the logic MITL over signals and signal automata—as well as the model-checking problem we tackle.

[2] These models are, however, not more expressive than signal automata.

Signals. An *interval* I is a non-empty convex subset of $\mathbb{R}_{\geq 0}$. If I is bounded ($\sup(I)$ exists), we write $|I|$ for $\sup(I) - \inf(I)$. Let AP be a finite set of atomic propositions. A *state* σ over AP is a subset of AP, i.e. $\sigma \in 2^{\mathsf{AP}}$. A *signal* γ over 2^{AP} is a function that maps each $t \in \mathbb{R}_{\geq 0}$ to a state over AP. Throughout this work, we restrict ourselves to signals that are *finitely variable*, i.e. the number of discontinuities is finite in each bounded interval. We rely on *timed state sequences* to represent signals. Intuitively, a timed state sequence partitions the reals into a sequence of contiguous time intervals during which the state remains constant. A *state sequence* $\overline{\sigma} = \sigma_0\sigma_1\sigma_2\cdots$ over 2^{AP} is an infinite sequence of states $\sigma_i \in 2^{\mathsf{AP}}$. An *interval sequence* $\overline{I} = I_0I_1I_2\cdots$ is an infinite sequence of intervals such that: **1.** for all $i \geq 0$, I_i and I_{i+1} are adjacent, i.e. $\sup(I_i) = \inf(I_{i+1})$ and $I_i \cap I_{i+1} = \emptyset$; **2.** for each $t \in \mathbb{R}_{\geq 0}$, we have $t \in I_i$ for some $i \geq 0$. An interval sequence is said *bipartite* if it alternates between singular and open intervals, i.e. I_i is singular for all even $i \geq 0$. Then, a *timed state sequence* over 2^{AP} is a pair $\kappa = (\overline{\sigma}, \overline{I})$ where $\overline{\sigma}$ is a state sequence over 2^{AP} and $\overline{I}$ is an interval sequence. We let $\kappa(t) = \sigma_i$ if $t \in I_i$ for some $i \geq 0$. We write $[\![\gamma]\!]$ (respectively, $[\![\gamma]\!]^{bp}$) for the set of all timed state sequences (respectively, timed state sequences with bipartite interval sequences) κ such that $\kappa(t) = \gamma(t)$ for all $t \in \mathbb{R}_{\geq 0}$.

Metric Interval Temporal Logic (MITL). We consider the satisfiability and model-checking problems for *Metric Interval Temporal Logic* (MITL), a real-time extension of Linear Temporal Logic (LTL), allowing temporal operators to be labelled with *non-singular* intervals. Formally, MITL formulae over AP are generated by the grammar:

$$\varphi := \top \mid p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi\, \mathcal{U}_I\, \varphi,$$

where $p \in \mathsf{AP}$ and I is a non-singular interval with endpoints in $\mathbb{N}_{\geq 0} \cup \{\infty\}$ (I is assumed to be $(0, \infty)$ when omitted).

In this work, we focus on the *continuous semantics* for MITL, in which formulae are interpreted over signals. Given a signal γ over 2^{AP}, $t \in \mathbb{R}_{\geq 0}$, and an MITL formula φ, the satisfaction relation $\gamma, t \models \varphi$ is defined as follows (following [4], we adopt the *strict-future* semantics for the temporal operators):

- $\gamma, t \models \top$;
- $\gamma, t \models p$ if $p \in \gamma(t)$;
- $\gamma, t \models \varphi_1 \wedge \varphi_2$ if $\gamma, t \models \varphi_1$ and $\gamma, t \models \varphi_2$;
- $\gamma, t \models \neg\varphi$ if $\gamma, t \not\models \varphi$;
- $\gamma, t \models \varphi_1\, \mathcal{U}_I\, \varphi_2$ if there exists $t' > t$ such that $t' - t \in I$, $\gamma, t' \models \varphi_2$ and $\gamma, t'' \models \varphi_1$ for all $t'' \in (t, t')$.

We write $\mathcal{S}(\varphi)$ for the set of all signals γ such that $\gamma \models \varphi$.

We will use standard syntactic sugar, e.g. $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\bot \equiv \neg\top$, $\varphi_1 \Rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, the 'eventually' operator $\mathbf{F}_I\varphi \equiv \top\, \mathcal{U}_I\, \varphi$, the 'globally' operator $\mathbf{G}_I\varphi \equiv \neg\mathbf{F}_I\neg\varphi$, and the 'release' operator $\varphi_1\, \mathcal{R}_I\, \varphi_2 \equiv \neg((\neg\varphi_1)\, \mathcal{U}_I\, (\neg\varphi_2))$. Hence, the semantics of the release operator can be defined as follows:

- $\gamma, t \models \varphi_1\, \mathcal{R}_I\, \varphi_2$ if for all $t' > t$ such that $t' - t \in I$, $\gamma, t' \models \varphi_2$ or there exists $t'' \in (t, t')$ such that $\gamma, t'' \models \varphi_1$.

In particular, we can make use of these operators to transform every formula φ into its *negative normal form* where the negations are pushed inwards so that they range on atomic propositions only.

Signal automata. Our tool support for MITL will be based on automata. We first give a formal definition of signal automata, and we will also present the classical timed automata afterwards. Like [4], we equip these automata with generalised Büchi acceptance conditions. From now on, a *propositional constraint* ϕ over AP is a set of states over AP; that we denote by means of a Boolean formula over AP. For example, assuming $\mathsf{AP} = \{p, q, r\}$, the propositional constraint $p \wedge \neg q$ denotes $\{\{p, r\}, \{p\}\}$. Let X be a finite set of clocks. The set $\mathcal{G}(X)$ of *clock constraints* g over X is generated by the grammar $g := \top \mid \bot \mid g \wedge g \mid x \bowtie c$ where $\bowtie \in \{\leq, <, \geq, >\}$, $x \in X$ and $c \in \mathbb{N}$. A *valuation* v of X is a mapping $v \colon X \to \mathbb{R}_{\geq 0}$. We denote by $\mathbf{0}$ the valuation that maps every clock to 0. The satisfaction of a constraint g by a valuation v is defined in the usual way and noted $v \models g$. For $t \in \mathbb{R}_{\geq 0}$, let $v + t$ be the valuation defined by $(v + t)(x) = v(x) + t$ for all $x \in X$. For $\lambda \subseteq X$, let $v[\lambda \leftarrow 0]$ be the valuation defined by $(v[\lambda \leftarrow 0])(x) = 0$ if $x \in \lambda$, and $(v[\lambda \leftarrow 0])(x) = v(x)$ otherwise.

▶ **Definition 1.** A *signal automaton* (SA) over 2^{AP} is a tuple $\mathcal{A} = (L, L_0, \alpha, X, \beta, \Delta, \mathcal{F})$ where

- L is a finite set of locations;
- $L_0 \subseteq L$ is the set of initial locations;
- α is the location labelling function that assigns to each location $\ell \in L$ a propositional constraint $\alpha(\ell) \subseteq 2^{\mathsf{AP}}$;
- X is a finite set of clocks;
- β is the location labelling function that assigns to each location $\ell \in L$ a clock constraint $\beta(\ell) \in \mathcal{G}(X)$;
- $\Delta \subseteq L \times 2^X \times L$ is the set of transitions where each transition consists of the source location, the clocks to be reset with this transition, and the target location;
- $\mathcal{F} \subseteq 2^L$ is the family of sets of accepting locations.

A *run* π of $\mathcal{A}$ on a signal γ over 2^{AP} is an infinite sequence of the following form:

$$\xrightarrow[v_0]{} (\ell_0, I_0) \xrightarrow[v_1]{\lambda_1} (\ell_1, I_1) \xrightarrow[v_2]{\lambda_2} (\ell_2, I_2) \xrightarrow[v_3]{\lambda_3} \dots$$

where: **1.** for all $i \geq 0$, ℓ_i is a locations of $\mathcal{A}$; **2.** the sequence $I_0 I_1 I_2 \cdots$ is an interval sequence; **3.** for all $i \geq 0$: $\lambda_i \subseteq X$; **4.** for all $i \geq 0$: v_i is a valuation of X; and that satisfies the following:

- [*Initiality*] $\ell_0 \in L_0$ and $v_0 = \mathbf{0}$; and
- [*Consecution*] For all $i \geq 0$: $(\ell_i, \lambda_{i+1}, \ell_{i+1}) \in \Delta$ and $v_{i+1} = (v_i + |I_i|)[\lambda_{i+1} \leftarrow 0]$; and
- [*Timing*] $v_\pi(t) \models \beta(\ell_\pi(t))$ for all $t \geq 0$, assuming $v_\pi(t) = v_i + (t - \sup(I_i))$ and $\ell_\pi(t) = \ell_i$ if $t \in I_i$ for some $i \geq 0$; and
- [*Adequation*] $\gamma(t) \in \alpha(\ell_\pi(t))$ for all $t \geq 0$.

We say that π is *bipartite* if $I_0 I_1 I_2 \cdots$ is bipartite. We say that π is *accepting* if for all $F \in \mathcal{F}$: $\{i \mid \ell_i \in F\}$ is infinite. A signal γ is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on γ. We write $\mathcal{S}(\mathcal{A})$ for the set of signals accepted by $\mathcal{A}$. For two SAs $\mathcal{A}_1$ and $\mathcal{A}_2$, we denote by $\mathcal{A}_1 \times \mathcal{A}_2$ their (asynchronous) product, defined in a manner similar to [4]: intuitively, in each location of this product, we can either fire only a transition of $\mathcal{A}_1$ (provided that that the guard in the current location of $\mathcal{A}_2$ is consistent with the one in destination of the transition), or only a transition of $\mathcal{A}_2$, or fire a transition in both signal automata in case their guards need to evolve synchronously. In particular, we have $\mathcal{S}(\mathcal{A}_1 \times \mathcal{A}_2) = \mathcal{S}(\mathcal{A}_1) \cap \mathcal{S}(\mathcal{A}_2)$.

We focus on the class of bipartite SA whose runs are bipartite by construction. An SA $\mathcal{A} = (L, L_0, \alpha, X, \beta, \Delta, \mathcal{F})$ is *bipartite* if there exists a partition of L into L^{sing}, L^{open} respecting the conditions given hereinafter. Intuitively, when reading a bipartite signal, $\mathcal{A}$ is in a location of L^{sing} (L^{open}) when it traverses a singular (respectively open) interval of γ:

- $L_0 \subseteq L^{sing}$;
- if $(\ell_1, \lambda, \ell_2) \in \Delta$ then $\ell_1 \in L^{sing}$ if and only if $\ell_2 \in L^{open}$;
- for each $\ell \in L_0$, $\beta(\ell)$ has $x = 0$ as a conjunct for some clock $x \in X$;
- if $(\ell_1, \lambda, \ell_2) \in \Delta$ with $\ell_1 \in L^{open}$ (and thus $\ell_2 \in L^{sing}$), then there is a clock $x \in X$ such that $x \in \lambda$ and $\beta(\ell_2)$ has $x = 0$ as a conjunct.

In the rest of the paper, we will assume that all SAs are bipartite.[3] There is no loss of generality, thanks to the following proposition [4] (see Appendix A for a proof):

▶ **Proposition 2.** *Every* SA $\mathcal{A}$ *can be turned into a bipartite* SA $\mathcal{A}^{bp}$ *such that* $\mathcal{S}(\mathcal{A}) = \mathcal{S}(\mathcal{A}^{bp})$.

From now on, when depicting bipartite SA, we use rectangle and rounded rectangles for the locations from L^{sing} and L^{open} respectively. Figure 1 shows an example of bipartite SA.

Satisfiability and model-checking problems. In this work, we consider two classical problems: satisfiability and model-checking of MITL. The *satisfiability problem* asks, given an MITL formula ϕ, whether $\mathcal{S}(\phi) \neq \emptyset$ (if it is the case, we say that ϕ is satisfiable). The *model-checking problem* asks, given an SA $\mathcal{A}$ and an MITL formula ϕ whether $\mathcal{S}(\mathcal{A}) \subseteq \mathcal{S}(\varphi)$. If it is the case, we write $\mathcal{A} \models \varphi$.

3 From MITL to signal automata

Our approach to MITL model-checking over signals is based upon a compositional translation from MITL to signal automata. The core idea is similar to the translation for the pointwise semantics reported in our previous work [15]: we keep track of the satisfiability of each temporal subformula (i.e. a subformula whose outermost operator is temporal) χ with an SA $\mathcal{C}_\chi$. From now on, we fix a set AP of atomic propositions and a *negative normal form* MITL formula φ over AP. To simplify the exposition, we restrict ourselves to a fragment of MITL in which only untimed and upper-bound operators are allowed, i.e. each bounding interval I is either $(0, \infty)$ or $(0, a)$, or $(0, a]$ for some positive integer a. This fragment, however, is already expressively complete for the full MITL [28, 37]. We further assume that each temporal subformula χ of φ appears only once in φ.

Triggers. Let Φ be the set of temporal subformulae of φ. We introduce a new atomic proposition p_χ for each subformula $\chi \in \Phi$ and we let $\mathsf{AP}_\Phi = \{p_\chi \mid \chi \in \Phi\}$. Each p_χ is called a *trigger* (for χ). Intuitively, *pulling* the trigger p_χ (i.e. setting p_χ to true) at some point means that χ is required to hold at that point. On the other hand, p_χ being false at some point does not mean that χ must not hold at that point—its satisfaction is simply not required there. The point of the triggers is to enable communication between the different component automata: when a formula φ is a subformula of ψ, the component SA $\mathcal{C}_\psi$ will pull the trigger of φ whenever the satisfaction of φ is needed to check the value of ψ. A key point of our construction is to avoid unnecessary pulling of triggers, in order to reduce the number of behaviours of the product automaton and mitigate the state explosion problem during the model checking phase. This is the point of the formulae $\overline{\psi}$, $*\psi$, $\sim\psi$ and $\widehat{\psi}$ that we introduce hereinafter. Concretely, the outcome of our construction for an MITL formula φ is a network of SA that accepts an AP_ϕ-decorated version of $\mathcal{S}(\varphi)$. In other words, the signals accepted our construction are over $\mathsf{AP} \cup \mathsf{AP}_\phi$ and their projections on AP yields $\mathcal{S}(\varphi)$, as stated in Theorem 8 at the end of the section.

[3] Note that a product of bipartite SAs is a bipartite SA.

For each (not necessarily temporal) subformula ψ of ϕ, we denote by $\mathcal{P}_\psi$ the set of atomic propositions $p_\chi \in \mathsf{AP}_\Phi$ such that χ is a top-level temporal subformula of ψ, i.e. the outermost operator of χ is $\mathcal{U}_I$ or $\mathcal{R}_I$, yet χ does not occur under the scope of another $\mathcal{U}_I$ or $\mathcal{R}_I$ in ψ. For instance, $\mathcal{P}_{p\mathcal{U}_I q \vee r\mathcal{U}_I(s\mathcal{R}t)} = \{p_{p\mathcal{U}_I q}, p_{r\mathcal{U}_I(s\mathcal{R}t)}\}$. For a signal γ' over $2^{\mathsf{P}'}$ (where P' is a set of atomic propositions) and $\mathsf{P} \subseteq \mathsf{P}'$, we denote by $\mathsf{proj}_{\mathsf{P}}(\gamma')$ the *projection* of γ' onto P, i.e. the signal obtained from γ' by hiding all the atomic propositions $p \notin \mathsf{P}$. For a set of signals $\mathcal{S}$ over $2^{\mathsf{P}'}$ and $\mathsf{P} \subseteq \mathsf{P}'$, we write $\mathsf{proj}_{\mathsf{P}}(\mathcal{S}) = \{\mathsf{proj}_{\mathsf{P}}(\gamma') \mid \gamma' \in \mathcal{S}\}$. Conversely, we say a signal γ' over $2^{\mathsf{P}'}$ *extends* a signal γ over 2^{P} ($\mathsf{P} \subseteq \mathsf{P}'$) if $\mathsf{proj}_{\mathsf{P}}(\gamma') = \gamma$.

Formulae over $\mathsf{AP} \cup \mathsf{AP}_\Phi$. We define some syntactic operations on Boolean combinations over $\mathsf{AP} \cup \mathsf{AP}_\Phi$ that will be used in the components that we describe later. Specifically, for a subformula ψ of φ, we define formulae $\overline{\psi}$ (that introduces the trigger variables in subformulae), $*\psi$ (that ensures that we do not pull any trigger of ψ), $\sim\psi$ (that checks that ψ does not hold, while none of its triggers are pulled), and $\widehat{\psi}$ (that checks ψ while triggering a minimal set of triggers).

The formula $\overline{\psi}$ is obtained from ψ by replacing all top-level temporal subformulae with their corresponding triggers. Formally, $\overline{\psi}$ is defined inductively as follows (where $p \in \mathsf{AP}$):

$$\overline{\psi_1 \wedge \psi_2} = \overline{\psi_1} \wedge \overline{\psi_2} \qquad \overline{\psi} = \psi \text{ when } \psi \text{ is } \top \text{ or } \bot \text{ or } p \text{ or } \neg p$$
$$\overline{\psi_1 \vee \psi_2} = \overline{\psi_1} \vee \overline{\psi_2} \qquad \overline{\psi} = p_\psi \text{ when } \psi \text{ is } \psi_1 \,\mathcal{U}_I\, \psi_2 \text{ or } \psi_1 \,\mathcal{R}_I\, \psi_2\,.$$

The formula $*\psi$, read as "do not pull the triggers of ψ", is used to ensure that our components only follow the 'minimal models' of ψ. It is defined as the conjunction of the negations of all $p_\chi \in \mathcal{P}_\psi$. As a concrete example,

$$*((\neg p \vee \psi_1 \,\mathcal{U}\, \psi_2) \wedge (q \vee \psi_3 \,\mathcal{R}\, (\psi_4 \,\mathcal{U}\, \psi_5))) = \neg p_{\psi_1 \mathcal{U} \psi_2} \wedge \neg p_{\psi_3 \mathcal{R} (\psi_4 \mathcal{U} \psi_5)}.$$

The formula $\sim\psi$ asserts that $\overline{\psi}$ is false and none of its triggers are pulled: $\sim\psi = \neg\overline{\psi} \wedge *\psi$. Finally, the formula $\widehat{\psi}$ is defined as $\mathsf{mm}(\overline{\psi})$ where $\mathsf{mm}(\phi)$ is defined inductively as follows:

$$\mathsf{mm}(\top) = \top \qquad \mathsf{mm}(\bot) = \bot \qquad \mathsf{mm}(p) = p \qquad \mathsf{mm}(\neg p) = \neg p$$
$$\mathsf{mm}(\phi_1 \vee \phi_2) = \big(\mathsf{mm}(\phi_1) \wedge \sim\phi_2\big) \vee \big(\mathsf{mm}(\phi_2) \wedge \sim\phi_1\big) \vee \big((\phi_1 \wedge \phi_2) \wedge *\phi_1 \wedge *\phi_2\big)$$
$$\mathsf{mm}(\phi_1 \wedge \phi_2) = \mathsf{mm}(\phi_1) \wedge \mathsf{mm}(\phi_2)\,.$$

First of all, we notice that formulae $\overline{\psi}$ and $\widehat{\psi}$ are equivalent, once we have projected away the propositions that are not in AP, in the following sense:

▶ **Proposition 3.** *For a subformula ψ of φ, if $\sigma \models \overline{\psi}$ for some state σ over $\mathsf{AP} \cup \mathcal{P}_\psi$, there is a state σ' over $\mathsf{AP} \cup \mathcal{P}_\psi$ such that $\sigma' \models \widehat{\psi}$ and $\mathsf{proj}_{\mathsf{AP}}(\sigma) = \mathsf{proj}_{\mathsf{AP}}(\sigma')$ (and vice versa).*

Proof. By induction on the structure of $\overline{\psi}$. For the direct implication, if $\overline{\psi} = \overline{\psi_1} \vee \overline{\psi_2}$ then one of the following must hold:

- $\sigma \models \overline{\psi_1}$ and $\sigma \not\models \overline{\psi_2}$: apply the induction hypothesis on $\sigma \setminus \mathcal{P}_{\psi_2}$ and $\overline{\psi_1}$ (note that $\mathcal{P}_{\psi_1} \cap \mathcal{P}_{\psi_2} = \emptyset$, and ψ_2 is in negative normal form).
- $\sigma \not\models \overline{\psi_1}$ and $\sigma \models \overline{\psi_2}$: apply the induction hypothesis on $\sigma \setminus \mathcal{P}_{\psi_1}$ and $\overline{\psi_2}$.
- $\sigma \models \overline{\psi_1}$ and $\sigma \models \overline{\psi_2}$: If $\sigma \setminus \mathcal{P}_{\psi_2} \not\models \overline{\psi_2}$, apply the induction hypothesis on $\sigma \setminus \mathcal{P}_{\psi_2}$ and $\overline{\psi_1}$. Otherwise if $\sigma \setminus \mathcal{P}_{\psi_1} \not\models \overline{\psi_1}$, apply the induction hypothesis on $\sigma \setminus \mathcal{P}_{\psi_1}$ and $\overline{\psi_2}$. Otherwise let $\sigma' = \sigma \setminus (\mathcal{P}_{\psi_1} \cup \mathcal{P}_{\psi_2})$.

The other cases of $\overline{\psi}$ are immediate. The other implication of the proof is simpler. ◀

Minimality of triggers. The real impact of $\widehat{\psi}$ with respect to ψ is to ensure the minimality of triggers pulled during an execution. Indeed, we now show that if φ is satisfied by a signal γ (over 2^{AP}), then there must be a way to extend γ into a signal γ' over $2^{\mathsf{AP} \cup \mathsf{AP}_\Phi}$ such that the triggers AP_Φ are only pulled when necessary in γ', and vice versa. This will be crucial to make our approach efficient in practice, as it reduces the behaviours of the product SA that accepts the whole formula φ. This observation is formalised in the following two propositions.

▶ **Proposition 4.** *For a signal γ over 2^{AP}, we have $\gamma, 0 \models \varphi$ if and only if there exists a signal γ' over $2^{\mathsf{AP} \cup \mathcal{P}_\varphi}$ extending γ such that $\gamma', 0 \models \widehat{\varphi}$, and for all $\chi \in \mathcal{P}_\varphi$ and $t \in \mathbb{R}_{\geq 0}$, $\mathsf{proj}_{\mathsf{AP} \cup \{p_\chi\}}(\gamma'), t \models (p_\chi \Rightarrow \chi)$.*

Proof. For the direct implication, let ζ be a signal over $2^{\mathsf{AP} \cup \mathcal{P}_\varphi}$ extending γ such that $p_\chi \in \zeta(t)$ if and only if $\gamma, t \models \chi$ for each $\chi \in \mathcal{P}_\varphi$ and $t \in \mathbb{R}_{\geq 0}$ (note that ζ is necessarily finitely-variable as γ is finitely-variable [4]). If $\zeta, 0 \models \widehat{\varphi}$, simply let $\gamma' = \zeta$ and we are done. If $\zeta, 0 \not\models \widehat{\varphi}$, apply Proposition 3 to $\zeta(0)$ and $\overline{\varphi}$ to obtain a state σ such that $\sigma \models \widehat{\varphi}$. Finally, let $\gamma'(0) = \sigma$ and $\gamma'(t) = \zeta(t) \setminus \mathcal{P}_\varphi$ for all $t \in \mathbb{R}_{>0}$. The other implication is immediate. ◀

▶ **Proposition 5.** *For a signal γ over $2^{\mathsf{AP} \cup \{p_\chi\}}$ where $\chi \in \Phi$ and either $\chi = \psi_1 \,\mathcal{U}_I\, \psi_2$ or $\chi = \psi_1 \,\mathcal{R}_I\, \psi_2$, we have $\gamma, t \models (p_\chi \Rightarrow \chi)$ for all $t \in \mathbb{R}_{\geq 0}$ if and only if there exists a signal γ' over $2^{\mathsf{AP} \cup \{p_\chi\} \cup \mathcal{P}_{\psi_1} \cup \mathcal{P}_{\psi_2}}$ extending γ such that*

- *if $\chi = \psi_1 \,\mathcal{U}_I\, \psi_2$ then, for each $t \in \mathbb{R}_{\geq 0}$, $\gamma', t \models p_\chi \Rightarrow \mathsf{Expand}_\chi$ with*

$$\begin{aligned}\mathsf{Expand}_\chi = \quad & \Big[(\widehat{\psi_1} \wedge \sim \psi_2)\,\mathcal{U}_I\,(* \psi_1 \wedge \widehat{\psi_2})\Big] \vee \Big[(\widehat{\psi_1} \wedge \widehat{\psi_2})\,\mathcal{U}\,\top\Big] \\ & \vee \Big[\mathbf{F}_I \widehat{\psi_2} \wedge \big(\widehat{\psi_1} \wedge \sim \psi_2\big)\,\mathcal{U}\,\big(\widehat{\psi_1} \wedge \sim \psi_2 \wedge (\widehat{\psi_1} \wedge \widehat{\psi_2})\,\mathcal{U}\,\top\big)\Big]\end{aligned}$$

- *if $\chi = \psi_1 \,\mathcal{R}_I\, \psi_2$ then, for each $t \in \mathbb{R}_{\geq 0}$, $\gamma', t \models p_\chi \Rightarrow \mathsf{Expand}_\chi$ with*

$$\begin{aligned}\mathsf{Expand}_\chi = \quad & \Big[(\sim \psi_1 \wedge \widehat{\psi_2})\,\mathcal{U}_I\,(\widehat{\psi_1} \wedge \widehat{\psi_2})\Big] \vee \Big[(\widehat{\psi_1} \wedge * \psi_2)\,\mathcal{U}\,\top\Big] \vee \Big[\mathbf{G}_I(\sim \psi_1 \wedge \widehat{\psi_2})\Big] \\ & \vee \Big[\mathbf{F}_I \widehat{\psi_1} \wedge \big(\sim \psi_1 \wedge \widehat{\psi_2}\big)\,\mathcal{U}\,\big(\sim \psi_1 \wedge \widehat{\psi_2} \wedge (\widehat{\psi_1} \wedge * \psi_2)\,\mathcal{U}\,\top\big)\Big]\end{aligned}$$

- *for each $p_\theta \in \mathcal{P}_{\psi_1} \cup \mathcal{P}_{\psi_2}$, we have $\mathsf{proj}_{\mathsf{AP} \cup \{p_\theta\}}(\gamma'), t \models (p_\theta \Rightarrow \theta)$ for all $t \in \mathbb{R}_{\geq 0}$.*

Proof. Assume that $\chi = \psi_1 \,\mathcal{U}_I\, \psi_2$ and let ζ be a signal over $2^{\mathsf{AP} \cup \{p_\chi\} \cup \mathcal{P}_{\psi_1} \cup \mathcal{P}_{\psi_2}}$ extending γ such that $p_\theta \in \zeta(t)$ if and only if $\gamma, t \models \theta$ for each $p_\theta \in \mathcal{P}_{\psi_1} \cup \mathcal{P}_{\psi_2}$ and $t \in \mathbb{R}_{\geq 0}$. For each $t \in \mathbb{R}_{\geq 0}$ such that $\gamma, t \models p_\chi$, since $\gamma, t \models \chi$ also holds, exactly one of the following must be true (note that $\inf(I) = 0$):

- there is $t' > t$, $t' - t \in I$ such that $\gamma, t' \models \psi_2$ and $\gamma, t'' \models \psi_1 \wedge \neg\psi_2$ for all $t'' \in (t, t')$;
- there is $t' > t$ such that $\gamma, t'' \models \psi_1 \wedge \psi_2$ for all $t'' \in (t, t')$;
- there are $t' > t$ and $t'' > t'$ such that in γ, $\psi_1 \wedge \neg\psi_2$ always holds in $(0, t']$ and $\psi_1 \wedge \psi_2$ always holds in (t', t'').

It follows that we can obtain a 'minimal labelling' from ζ via Proposition 3. More precisely, we apply Proposition 3 to constant segments of ζ and $\overline{\psi_1}$, $\overline{\psi_2}$, or both $\overline{\psi_1}$ and $\overline{\psi_2}$, as required by the interpretation of p_χ in γ. For example, in the first case above, $\gamma'(t'')$ for each $t'' \in (t, t')$ is obtained by applying Proposition 3 to $\zeta(t'') \setminus \mathcal{P}_{\psi_2}$ and $\overline{\psi_1}$; $\gamma'(t')$ is obtained by applying Proposition 3 to $\zeta(t') \setminus \mathcal{P}_{\psi_1}$ and $\overline{\psi_2}$. Similar arguments can be made for $\chi = \psi_1 \,\mathcal{R}\, \psi_2$. The other implication is simpler. ◀

▶ **Corollary 6.** *For a signal γ over 2^{AP}, we have $\gamma, 0 \models \varphi$ if and only if there exists a signal γ' over $2^{\mathsf{AP} \cup \mathsf{AP}_\Phi}$ extending γ such that $\gamma', 0 \models \widehat{\varphi}$, and for all $\chi \in \Phi$ and $t \in \mathbb{R}_{\geq 0}$, $\gamma', t \models p_\chi \Rightarrow \mathsf{Expand}_\chi$ (where Expand_χ is one of the formulae in Proposition 5).*

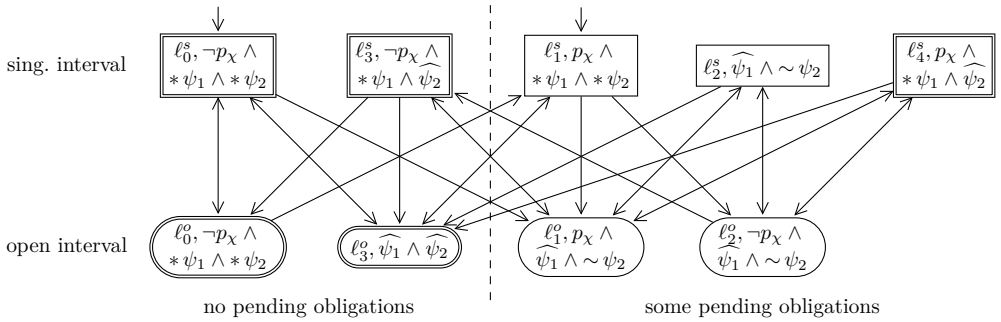

Figure 1 The component SA $\mathcal{C}_\chi$ for $\chi = \psi_1 \,\mathcal{U}\, \psi_2$.

The components. We are now ready to present the components $\mathcal{C}_\chi$ for $\chi \in \Phi$.

The component $\mathcal{C}_\chi$ for $\chi = \psi_1 \,\mathcal{U}\, \psi_2$ is given in Figure 1. We now explain how it has been produced. Thanks to Proposition 2, we provide a bipartite SA (in particular, we will read timed sequences with bipartite interval sequences only), where 'singular' locations are on top, and 'open' locations at the bottom. First, we focus on locations ℓ_0^s and ℓ_0^o, that are used as long as trigger p_χ is not pulled: then, there is no need to pull any trigger of ψ_1 nor ψ_2, which is ensured via the use of formula $* \psi_1 \wedge * \psi_2$. Consider then the first time when trigger p_χ is pulled (by another component automaton): it is either in a singular interval in which case we jump into location ℓ_1^s (this creates a pending obligation, since such an 'until' with our strict semantics cannot be fulfilled right away in a singular interval: this means, in particular, that we do not need to pull any trigger for ψ_1 or ψ_2, thus checking $* \psi_1 \wedge * \psi_2$), or in an open interval in which case we jump either into location ℓ_1^o if ψ_2 does not hold (i.e. if $\sim \psi_2$ holds), or into location ℓ_3^o if ψ_2 holds (i.e. if $\widehat{\psi_2}$ is in the guard) which fulfils right away the new obligation (notice that, in the figure, we did not put p_χ in the guard of this location, for simplification: we will discuss this point more in detail afterwards).

When p_χ is first pulled in an open interval (which means we jump into location ℓ_1^o or ℓ_3^o), by the semantics of the 'until' operator, ψ_1 must also hold in that interval. When in ℓ_3^o, the successors are the same as in ℓ_0^o. When in ℓ_1^o with a pending obligation, there are two cases for the next jump:

- either ψ_2 holds in the next singular interval, and then no trigger of ψ_1 needs to be pulled (i.e. guard $* \psi_1 \wedge \widehat{\psi_2}$): if there are no new pulled trigger p_χ, we jump into location ℓ_3^s; otherwise, we jump into location ℓ_4^s where we still have a new pending obligation, but the location is still made accepting to record the fact that the previous obligation has been fulfilled.
- or ψ_2 does not hold, in which case ψ_1 should hold (i.e. guard $\widehat{\psi_1} \wedge \sim \psi_2$): we then jump into location ℓ_2^s whether or not a new trigger p_χ is pulled.

When p_χ is first pulled in a singular interval (which means we jump into location ℓ_1^s), there is no need to pull any trigger of ψ_1 nor ψ_2. Then, while in one of the 'singular' locations ℓ_1^s, ℓ_2^s or ℓ_4^s, with a pending obligation, in the next jump, there are two cases:

- either ψ_2 holds in the next open interval, in which case ψ_1 should still hold (because of the semantics of the 'until' operator): we can jump into the previously introduced location ℓ_3^o.
- or ψ_2 does not hold (then, ψ_1 should hold anyway) and we jump either in location ℓ_1^o if a new trigger p_χ is pulled, or in ℓ_2^o is no new trigger p_χ is pulled. Location ℓ_2^o has the

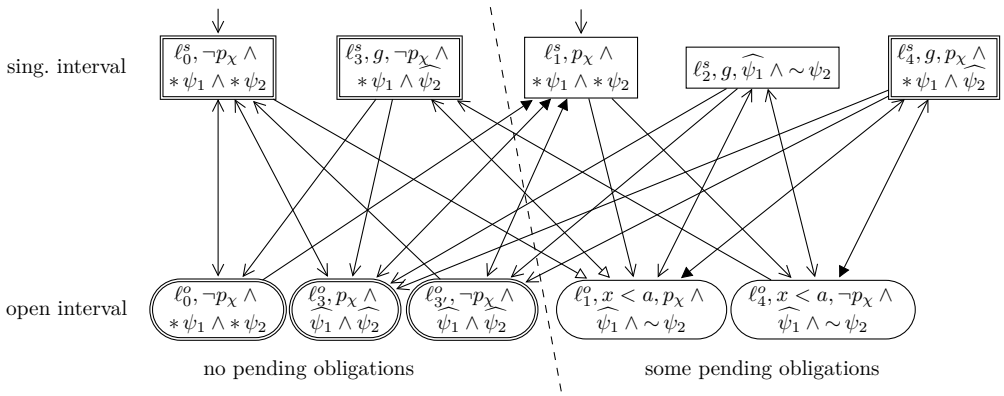

Figure 2 The component SA $\mathcal{C}_\chi$ for $\psi_1 \mathcal{U}_{(0,a)} \psi_2$. We use a Boolean variable `si` to signify whether the oldest pending obligation has been pulled in a singular interval or not. The transitions with ▶ or ▷ reset x; the ones with ▶ (resp. ▷) set `si` to true (resp. false). The clock constraint g is defined as $(\texttt{si} \wedge x < a) \vee (\neg\texttt{si} \wedge x \leq a)$.

same successors as ℓ_1^o but we still need to distinguish them since ℓ_1^o must check that a new pending obligation is pulled.

Initially, we do not want to pull any trigger of ψ_1 or ψ_2, therefore, ℓ_0^s and ℓ_1^s are the two initial locations, depending on whether trigger p_χ is initially pulled or not. Accepting locations are the one where either there are no more pending obligations, or a pending obligation has been fulfilled while a new trigger is being pulled (location ℓ_4^s).

Notice that, thanks to the use of $*\psi_i$ and $\widehat{\psi_i}$ formulas, only the necessary triggers in $\mathcal{P}_{\psi_1} \cup \mathcal{P}_{\psi_2}$ are pulled during an execution of this component. Indeed, this is not true for location ℓ_3^o: when going from locations ℓ_0^s or ℓ_3^s, to pull only minimal sets of triggers, we must make sure to go in ℓ_3^o only when a new trigger p_χ is pulled. This requires to split this location into two (one where p_χ holds, the other where it does not). For simplicity, we did not do it in the figure, but we apply this splitting in the next component we present.

This next component $\mathcal{C}_\chi$ is the one for $\chi = \psi_1 \mathcal{U}_{(0,a)} \psi_2$ (Figure 2), that is obtained by adding a clock x and suitable clock constraints. Intuitively, it suffices to use only one clock because for $I = (0, a)$, all new obligations are implied by the oldest pending obligation. This means that the clock should be reset when entering in a location where a trigger is pulled while all the previous obligations have been fulfilled: this is a priori the case when entering in locations ℓ_1^s, ℓ_1^o, and ℓ_4^s from locations $\{\ell_0^s, \ell_3^s, \ell_0^o, \ell_3^o, \ell_{3'}^o\}$. Now, the valuation of x would fix a deadline for the satisfaction of ψ_2. Indeed, as long as ψ_2 does not hold, we must check that $x < a$. When ψ_2 is next fulfilled, we also check that $x < a$. However, this is not correct for two reasons.

First, when checking the requirements $x < a$, this is not correct if the oldest pending obligations appeared in an open interval: indeed, it is still correct to fulfil ψ_2 in a singular interval where $x = a$. This requires that we register, when resetting clock x, if the trigger is pulled in a singular interval or not. To ease the presentation, we use a Boolean variable `si` to record that the trigger has been pulled in a singular interval. Pictorially, we use transitions with ▶ heads to reset the clock x and setting `si` to true, while transitions with ▷ heads reset clock x and set `si` to false. Then, the clock constraint that must be checked in singular interval (whether or not ψ_2 is currently fulfilled) is not $x < a$ but g defined by $(\texttt{si} \wedge x < a) \vee (\neg\texttt{si} \wedge x \leq a)$: in particular, the guard g in location ℓ_2^s models the fact

that if the oldest obligation has been triggered in an open interval (`si` is false), it is not a contradiction to not yet fulfil ψ_2 at time $x = a$, but then, the only fireable transitions are the one towards ℓ_3^o and $\ell_{3'}^o$ where ψ_2 then holds. This also explains why guard g does not need to be checked when entering in ℓ_3^o and $\ell_{3'}^o$.

Second, this cannot be done as such when entering location ℓ_4^s since the guard g must be checked *before* resetting clock x that records the deadline of the next pending obligation. Indeed, we simply delay the reset and modification of variable `si` to the next transition towards ℓ_1^o or ℓ_4^o.

The component for $\psi_1 \mathcal{U}_{(0,a]} \psi_2$ is similar and hence omitted. The components for 'release' operators follow the same pattern as the ones for 'until'. Due to lack of place, we present them in Appendix B. Then:

▶ **Proposition 7.** *For each $\chi \in \Phi$, the component $\mathcal{C}_\chi$ accepts exactly all signals γ over $2^{\mathsf{AP} \cup \mathsf{AP}_\Phi}$ such that $\gamma, t \models p_\chi \Rightarrow \mathsf{Expand}_\chi$ for all $t \in \mathbb{R}_{\geq 0}$ (where Expand_χ is one of the formulae in Proposition 5).*

Finally, we need a simple initial component $\mathcal{C}_{init}$ which enforces $\widehat{\varphi}$ at $t = 0$ and $* \varphi$ at all $t > 0$, as suggested by Proposition 5. We can now state the main theorem of this section.

▶ **Theorem 8.** $\mathsf{proj}_{\mathsf{AP}} \left(\mathcal{S}(\mathcal{C}_{init} \times \prod_{\chi \in \Phi} \mathcal{C}_\chi) \right) = \mathcal{S}(\varphi)$.

4 From signal automata to timed automata

In this section, we provide a new approach to check the emptiness of *signal* automata that can be implemented by relying on existing tools for *timed* automata. To this end, we explain how to encode an SA $\mathcal{A}$ into a timed automaton $\mathcal{B}^{\mathcal{A}}$ that accepts exactly the 'time words' counterparts of the signals accepted by $\mathcal{A}$. Moreover, the construction can be used in a compositional manner: if $\mathcal{A}$ is the product of a number of component SAs, $\mathcal{B}^{\mathcal{A}}$ can be obtained as the product of the TAs that result from applying the construction to the components of $\mathcal{A}$. As the construction is *emptiness-preserving*, it can serve as a bridge between the MITL-to-SA translation in the previous section and existing TA-based tools. We start by recalling formally what are timed automata.

Timed words and timed automata. A *time sequence* is an infinite sequence $\overline{\tau} = \tau_0 \tau_1 \tau_2 \dots$ of *timestamps* such that **1.** $\tau_0 = 0$; **2.** for all $i \geq 0$, $\tau_i \leq \tau_{i+1}$; **3.** for all $t \in \mathbb{R}_{\geq 0}$, there is some $i \geq 0$ such that $\tau_i > t$. A *timed word* $\rho = (\overline{\sigma}, \overline{\tau})$ over 2^{AP} is a pair of a state sequence $\overline{\sigma}$ over 2^{AP} and a time sequence $\overline{\tau}$. Alternatively, we may see ρ as an infinite sequence $(\sigma_0, \tau_0)(\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots$ of *events* (σ_i, τ_i). We now define timed automata, with generalised acceptance conditions as before (used by [27] in the untimed setting).

▶ **Definition 9.** A *timed automaton* (TA) over 2^{AP} is a tuple $\mathcal{A} = (L, L_0, X, \Delta, \mathcal{F})$ where

- L is a finite set of locations;
- $L_0 \subseteq L$ is the set of initial locations;
- X is a finite set of clocks;
- $\Delta \subseteq L \times 2^{2^{\mathsf{AP}}} \times \mathcal{G}(X) \times 2^X \times L$ is the set of transitions;
- $\mathcal{F} \subseteq 2^L$ is the family of sets of accepting locations.

A *run* π of $\mathcal{A}$ on a timed word $\rho = (\sigma_0, \tau_0)(\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots$ over 2^{AP} is an infinite sequence

$$(\ell_0, v_0) \xrightarrow[\sigma_0, d_0]{\lambda_1} (\ell_1, v_1) \xrightarrow[\sigma_1, d_1]{\lambda_2} (\ell_2, v_2) \xrightarrow[\sigma_2, d_2]{\lambda_3} \cdots$$

where, for all $i \geq 0$: **1.** ℓ_i is a locations of $\mathcal{A}$; **2.** v_i is a valuation of X; **3.** $d_i = \tau_i - \tau_{i-1}$ (assuming $\tau_{-1} = 0$) **4.** $\lambda_i \subseteq X$; and that satisfies the following:

- [*Initiality*] $\ell_0 \in L_0$; and
- [*Consecution*] for all $i \geq 0$: $(\ell_i, \phi, g, \lambda_{i+1}, \ell_{i+1}) \in \Delta$ with $\sigma_i \in \phi$ and $v_i + d_i \models g$; and
- [*Timing*] for all $i \geq 0$, $v_{i+1} = (v_i + d_i)[\lambda_{i+1} \leftarrow 0]$.

We say that π is *accepting* if for all accepting sets $F \in \mathcal{F}$, the set $\{i \mid \ell_i \in F\}$ is infinite. A timed word ρ is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on ρ. We write $\mathcal{L}(\mathcal{A})$ for the set of timed words accepted by $\mathcal{A}$. For two TAs $\mathcal{A}_1$ and $\mathcal{A}_2$, we denote by $\mathcal{A}_1 \times \mathcal{A}_2$ their (synchronous) product [3]. In particular, we have $\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

Translation from SA to TA. We first explain how we map signals to timed words. To do so, we select a bipartite state sequence κ corresponding to γ, and we express the state changes along κ in a timed word. Formally, for a signal γ and a timed state sequence $\kappa = (\sigma_0, I_0)(\sigma_1, I_1)\cdots$ s.t. $\kappa \in [\![\gamma]\!]^{bp}$ (i.e., I_i is singular for all even $i \geq 0$), we define:

$$[\kappa]_{tw} = (\sigma_0, \sup(I_0))(\sigma_1, \inf(I_1))(\sigma_2, \sup(I_2))(\sigma_3, \inf(I_3))\cdots .$$

Note that we represent a state change at time t by two events with timestamp t (note that $\sup(I_i) = \inf(I_{i+1})$ for each even $i \geq 0$). Abusing notations, we write $[\gamma]_{tw} = \{[\kappa]_{tw} \mid \kappa \in [\![\gamma]\!]^{bp}\}$ and $[\mathcal{S}]_{tw} = \bigcup_{\gamma \in \mathcal{S}} [\gamma]_{tw}$ for a set $\mathcal{S}$ of signals.

▶ **Proposition 10.** *Given a (bipartite)* SA $\mathcal{A}$*, we can construct a* TA $\mathcal{B}^{\mathcal{A}}$ *such that* $\mathcal{L}(\mathcal{B}^{\mathcal{A}}) = [\mathcal{S}(\mathcal{A})]_{tw}$*. In particular, if* $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ *then* $\mathcal{L}(\mathcal{B}^{\mathcal{A}_1} \times \cdots \times \mathcal{B}^{\mathcal{A}_n}) = [\mathcal{S}(\mathcal{A})]_{tw}$.

Proof (sketch). For a clock constraint $g \in \mathcal{G}(X)$, let $g^{\leftarrow}$ be the clock constraint obtained from g by replacing all clauses of the form '$x \leq c$' with '$x < c$' and all '$x > c$' with '$x \geq c$'. Likewise, let $g^{\rightarrow}$ be the clock constraint obtained from g by replacing all '$x < c$' with '$x \leq c$' and all '$x \geq c$' with '$x > c$'. The following statements hold (for a valuation v of X):

- $v \models g^{\leftarrow}$ if and only if for some $\delta \in \mathbb{R}_{>0}$, we have $v + t \models g$ for all $t \in (0, \delta]$.
- $v \models g^{\rightarrow}$ if and only if for some $\delta \in \mathbb{R}_{>0}$, we have $v' \models g$ for all valuations v' of X such that $v' + t = v$ for some $t \in (0, \delta]$.

In what follows, we write $g[\lambda \leftarrow 0]$ for the clock constraint obtained from g by replacing all occurrences of clocks $x \in \lambda$ with 0. For $\mathcal{A} = (L, L_0, \alpha, X, \beta, \Delta, \mathcal{F})$ (which by assumption is bipartite and $L = L^{sing} \uplus L^{open}$), define $\mathcal{B} = (L^{\mathcal{A}}, L_0^{\mathcal{A}}, X^{\mathcal{A}}, \Delta^{\mathcal{A}}, \mathcal{F}^{\mathcal{A}})$ where

- $L^{\mathcal{A}} = \{\ell^{sing} \mid \ell \in L^{sing}\} \cup \{\dot{\ell}^{sing}, \ell^{open}, \dot{\ell}^{open} \mid \ell \in L^{open}\} \cup \{\ell^{init}\}$;
- $L_0^{\mathcal{A}} = \{\ell^{init}\}$;
- $X^{\mathcal{A}} = X \cup \{y\}$ where y is a fresh clock;
- $\Delta^{\mathcal{A}} = \quad \{(\ell^{init}, \alpha(\ell), \beta(\ell) \wedge y = 0, \emptyset, \ell^{sing}) \mid \ell \in L_0\}$
  $\cup \{(\ell_1^{sing}, \alpha(\ell_2), \beta(\ell_2)^{\leftarrow}[\lambda \leftarrow 0] \wedge y = 0, \lambda, \ell_2^{open}) \mid (\ell_1, \lambda, \ell_2) \in \Delta\}$
  $\cup \{(\ell_1^{open}, \alpha(\ell_2), \beta(\ell_1)^{\rightarrow} \wedge \beta(\ell_2)[\lambda \leftarrow 0] \wedge y > 0, \lambda \cup \{y\}, \ell_2^{sing}) \mid (\ell_1, \lambda, \ell_2) \in \Delta\}$
  $\cup \{(\ell^{open}, \alpha(\ell), \beta(\ell)^{\rightarrow} \wedge \beta(\ell)[\lambda \leftarrow 0] \wedge y > 0, \lambda \cup \{y\}, \dot{\ell}^{sing}) \mid \ell \in L^{open}\}$
  $\cup \{(\dot{\ell}^{sing}, \alpha(\ell), \beta(\ell)^{\leftarrow}[\lambda \leftarrow 0] \wedge y = 0, \lambda, \dot{\ell}^{open}) \mid \ell \in L^{open}\}$
  $\cup \{(\dot{\ell}^{open}, \alpha(\ell), \beta(\ell)^{\rightarrow} \wedge \beta(\ell)[\lambda \leftarrow 0] \wedge y > 0, \lambda \cup \{y\}, \dot{\ell}^{sing}) \mid \ell \in L^{open}\}$
  $\cup \{(\dot{\ell}_1^{open}, \alpha(\ell_2), \beta(\ell_1)^{\rightarrow} \wedge \beta(\ell_2)[\lambda \leftarrow 0] \wedge y > 0, \lambda \cup \{y\}, \ell_2^{sing}) \mid (\ell_1, \lambda, \ell_2) \in \Delta\}$
- $\mathcal{F}^{\mathcal{A}} = \{\{\ell^{sing} \mid \ell \in L^{sing} \cap F\} \cup \{\ell^{open} \mid \ell \in L^{open} \cap F\} \mid F \in \mathcal{F}\}$.

Intuitively, the 'dotted' locations $\dot{\ell}^{sing}$, $\dot{\ell}^{open}$ are used to allow interleaving and stuttering as $\mathcal{A}$ stays in $\ell \in L^{open}$: this is crucial to make the asynchronous product $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ and the synchronous product $\mathcal{B}^{\mathcal{A}_1} \times \cdots \times \mathcal{B}^{\mathcal{A}_n}$ match. Finally, for pragmatic reasons, we make

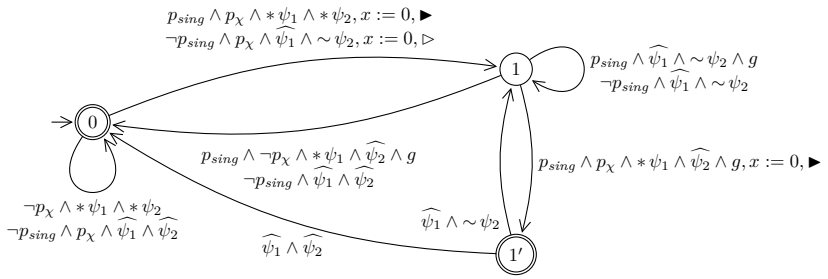

Figure 3 The component TA $\mathcal{B}_\chi$ for $\chi = \varphi_1 \,\mathcal{U}_{(0,a)}\, \varphi_2$. We use a Boolean variable `si` to signify whether the current p_χ-interval is left-closed. The transitions with $\blacktriangleright$ (respectively, $\triangleright$) set `si` to true (respectively, false). The clock constraint g is defined as $(\texttt{si} \wedge x < a) \vee (\neg\texttt{si} \wedge x \leq a)$.

suitable modifications to $\mathcal{B}$ to obtain a *strongly non-Zeno* TA $\mathcal{B}^{\mathcal{A}}$ (i.e. a TA in which time progresses), as in [23]. ◀

The proposition above works for any (bipartite) SA. For $\mathcal{C}_{init}$ or each component $\mathcal{C}_\chi$ ($\chi \in \Phi$) in the previous section, however, we can suppress all the 'dotted' locations $\dot{\ell}^{sing}$, $\dot{\ell}^{open}$ and build a much simpler TA (which we denote by $\mathcal{B}_{init}$ or $\mathcal{B}_\chi$, respectively).[4] Our main result can then be stated as the following theorem, where the projection operator proj is defined in a similar way as in the setting of signals.

▶ **Theorem 11.** $\mathsf{proj}_{\mathsf{AP}}\left(\mathcal{L}(\mathcal{B}^{\mathcal{A}} \times \mathcal{B}_{init} \times \prod_{\chi\in\Phi} \mathcal{B}_\chi)\right) = \mathsf{proj}_{\mathsf{AP}}\left(\left[\mathcal{S}(\mathcal{A} \times \mathcal{C}_{init} \times \prod_{\chi\in\Phi} \mathcal{C}_\chi)\right]_{tw}\right)$ *for any given* SA $\mathcal{A}$ *over* $2^{\mathsf{AP}\cup\mathsf{AP}_\Phi}$ *whose propositional constraints can be written as Boolean combinations over* AP *(i.e. do not involve atomic propositions in* AP_Φ*).*

As an example, the component TA $\mathcal{B}_\chi$ for $\chi = \psi_1 \,\mathcal{U}_{(0,a)}\, \psi_2$ (in which we use a new atomic proposition p_{sing} that holds on 'singular' transitions) is depicted in Figure 3.

5 Implementation and experiments

We have implemented the translation as an extension of our tool MightyL [15]. Given a formula φ over AP in MITL,[5] the tool generates the model TA $\mathcal{B}^{\mathcal{A}}$ where $\mathcal{A}$ is a universal SA over $2^{\mathsf{AP}\cup\mathsf{AP}_\Phi}$, the initial component TA $\mathcal{B}_{init}$, and the corresponding component TAs $\mathcal{B}_\chi$ for each temporal subformula χ of φ in the Uppaal xml format. The user can, of course, replace $\mathcal{B}^{\mathcal{A}}$ with the model TA $\mathcal{M}$ of their choice and perform model-checking with existing TA-based tools.[6] Our implementation is publicly available and can be executed directly on the webpage: `http://www.ulb.ac.be/di/verif/mightyl`. In the following experiments, we use LTSmin [30] (with opaal [34], which enables support for Uppaal xml files) as the back-end model checker and report its execution times (using only a single core) on a Pentium B970 (2.3GHz) machine with 6GB RAM running Ubuntu 17.04. We omit the execution times for MightyL as it is less than 0.1s on all our benchmarks.

[4] The product of these TAs corresponds to the *synchronous* product (in which interleaving and stuttering are disallowed [8]) of $\mathcal{C}_{init}$ and $\mathcal{C}_\chi$ for all $\chi \in \Phi$.

[5] More precisely, our tool accepts all temporal operators that are labelled with intervals of the form $(0,\infty)$, $[0,\infty)$, $(0,a)$, $[0,a)$, $(0,a]$ or $[0,a]$. If 0 is included in the interval, the temporal operator is given a *weak-future* interpretation [39], e.g. $\psi_1 \,\mathcal{U}^w_{[0,a)}\, \psi_2 \iff \psi_2 \vee (\psi_1 \wedge \psi_1 \,\mathcal{U}_{(0,a)}\, \psi_2)$. Remember that general MITL formulae can be rewritten into formulae of this fragment, e.g., $\mathbf{F}_{(a,\infty)}\psi \iff \mathbf{G}_{(0,a]}\mathbf{F}\psi$.

[6] We require $\mathcal{M}$ to be strongly non-Zeno and $\mathcal{L}(\mathcal{M}) = [\mathcal{S}(\mathcal{A})]_{tw}$ where $\mathcal{A}$ is an SA over $2^{\mathsf{AP}\cup\mathsf{AP}_\Phi}$ that satisfies the conditions in Theorem 11.

Table 1 Execution times for the 'parametric formulae' benchmark set. The columns 'Pointwise' correspond to the approach of [15] and the columns 'Continuous' correspond to the approach of this article (where OOM stands for out-of-memory). The three numbers of each entry correspond to the time taken by OPAAL to translate UPPAAL XML into C++, the time taken by the g++ compiler, and the actual model-checking time taken by LTSMIN, respectively.

Formula	Continuous	Pointwise	Formula	Continuous	Pointwise
$F(5,[0,\infty))$	0.43s/1.03s/0.32s	0.35s/0.98s/0.21s	$G(5,[0,\infty))$	0.45s/1.04s/0.39s	0.34s/0.99s/0.54s
$F(10,[0,\infty))$	0.74s/1.28s/0.31s	0.61s/1.20s/0.60s	$G(10,[0,\infty))$	0.78s/1.28s/39.43s	0.59s/1.18s/28.71s
$F(5,[0,5])$	0.62s/1.14s/0.16s	0.41s/1.01s/0.19s	$G(5,[0,5])$	1.11s/1.43s/0.56s	0.49s/1.06s/0.32s
$F(10,[0,5])$	1.16s/1.48s/0.41s	0.71s/1.26s/2.89s	$G(10,[0,5])$	OOM	0.89s/1.37s/14.75s
$F(2,(5,\infty))$	0.67s/1.15s/0.20s	0.21s/0.85s/0.09s	$G(2,(5,\infty))$	0.45s/1.05s/0.20s	0.20s/0.86s/0.09s
$F(5,(5,\infty))$	1.48s/1.69s/17.06s	0.36s/0.99s/0.21s	$G(5,(5,\infty))$	0.94s/1.41s/3.62s	0.35s/0.98s/0.31s
$F(10,(5,\infty))$	OOM	0.62s/1.20s/0.64s	$G(10,(5,\infty))$	OOM	0.61s/1.17s/45.67s
$U(5,[0,\infty))$	0.36s/0.98s/0.16s	0.31s/0.96s/0.11s	$R(5,[0,\infty))$	0.42s/1.04s/0.29s	0.30s/0.97s/0.25s
$U(10,[0,\infty))$	0.65s/1.22s/0.29s	0.57s/1.21s/0.50s	$R(10,[0,\infty))$	0.70s/1.28s/2.77s	0.54s/1.17s/9.71s
$U(5,[0,5])$	0.53s/1.10s/0.11s	0.34s/0.97s/0.07s	$R(5,[0,5])$	0.93s/1.31s/0.27s	0.42s/1.01s/0.11s
$U(10,[0,5])$	1.04s/1.42s/0.36s	0.66s/1.36s/0.53s	$R(10,[0,5])$	1.97s/1.94s/11.52s	0.83s/1.31s/18.83s
$U(2,(5,\infty))$	0.42s/0.97s/0.14s	0.18s/0.82s/0.05s	$R(2,(5,\infty))$	0.31s/0.92s/0.07s	0.17s/0.82s/0.05s
$U(5,(5,\infty))$	1.34s/1.60s/6.93s	0.32s/0.95s/0.09s	$R(5,(5,\infty))$	0.92s/1.37s/2.88s	0.30s/0.94s/0.27s
$U(10,(5,\infty))$	OOM	0.56s/1.17s/0.93s	$R(10,(5,\infty))$	OOM	0.54s/1.17s/12.45s

Satisfiability of parametric formulae. We consider the satisfiability of a set of parametric MITL formulae modified from [14, 24]. The goal of this benchmark set is to give a rough comparison between the performance of our approach in the *pointwise* semantics (the original aim of MIGHTYL; we refer the reader to [15,39] for more details) with that in the continuous semantics (this article). For $k \geq 2$ and an interval I, let:

$$F(k,I) = \textstyle\bigwedge_{i=1}^{k} \mathbf{F}_I^w p_i, \qquad G(k,I) = \textstyle\bigwedge_{i=1}^{k} \mathbf{G}_I^w p_i,$$
$$U(k,I) = (\dots(p_1 \,\mathcal{U}_I^w\, p_2)\,\mathcal{U}_I^w \dots)\,\mathcal{U}_I^w\, p_k, \qquad R(k,I) = (\dots(p_1 \,\mathcal{R}_I^w\, p_2)\,\mathcal{R}_I^w \dots)\,\mathcal{R}_I^w\, p_k,$$

where $\mathbf{F}_I^w$, $\mathbf{G}_I^w$, etc., are *weak-future* temporal operators [39]. The formulae in the benchmark set are given in Table 1. For the pointwise case, these are the actual formulae that we pass to MIGHTYL; for the continuous case, standard rewriting rules are applied to handle the lower-bound temporal operators (e.g. $\mathbf{F}_{(5,\infty)}p \iff \mathbf{G}_{(0,5]}\mathbf{F}p$).[7] From the execution times in Table 1, it is evident that OPAAL and g++ are not performance bottlenecks. For smaller formulae, the times taken by LTSMIN are very short. For larger formulae, however, as LTSMIN uses depth-first search for OPAAL-generated models, it sometimes goes very deep into the state space and results in out-of-memory.

Validity and redundancy of specifications. We say an MITL formula φ is *valid* if $\neg\varphi$ is not satisfiable. If φ is of the form $\bigwedge_{1\leq i\leq k}\varphi_i$, we say that the conjunct φ_i is *redundant* in φ if the formula $(\bigwedge_{\substack{1\leq j\leq k\\ j\neq i}}\varphi_j) \Rightarrow \varphi_i$ is valid. In [20], MITL specifications created by non-expert users are checked for satisfiability, validity and redundancy. We report the execution times of our approach on some of their checks in Table 2. To see the effect of forcing minimal triggers, we also give the execution times when this is not imposed. We also reproduce the execution times reported in [20] in the table; since we do not impose *a priori* bounds on state changes (as opposed to [20]) and we use a much less powerful CPU, these numbers are not meant for direct comparison but rather for reference.

[7] Of course, the resulting formulae are interpreted over signals, in contrast to their pointwise counterparts; but we expect the computational efforts needed to check their satisfiability to be similar.

Table 2 Execution times for the satisfiability, validity and redundancy checks in [20].

Formula	Our approach	Our approach w/o minimality	[20]
$\phi_1 = \mathbf{F}_{[0,30]}p_1 \wedge \mathbf{F}_{[0,20]}p_1$	5.95s	8.67s	14s
$\phi_2 = \mathbf{F}_{[0,30]}(p_1 \Rightarrow \mathbf{G}_{[0,20]}p_1)$	3.05s	5.3s	7s
$\phi_4 = \mathbf{G}_{[0,40]}p_1 \wedge \mathbf{G}_{[0,40]}\mathbf{F}_{[0,10]}p_1$	7.23s	52.43s	29s
$\phi_5 = \mathbf{F}_{[0,40]}(p_1 \vee p_3) \wedge \mathbf{F}_{[0,40]}p_2 \wedge \mathbf{F}_{[0,40]}\mathbf{G}_{[0,30]}p_1$	12.12s	>1200s	126s

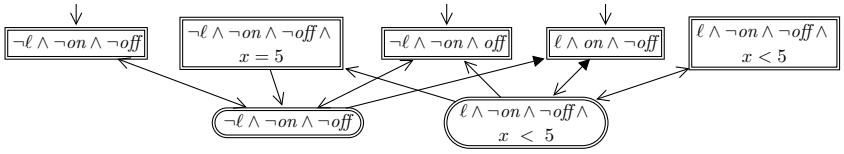

Figure 4 The SA $\mathcal{A}_{lamp}$. The transitions with solid tips reset clock x.

Model-checking a timed lamp. We consider a case study of a timed lamp from [10]. The lamp is controlled by two buttons 'on' and 'off', which can only be pressed instantaneously but not simultaneously. The buttons turn the lamp on and off as expected, and the lamp turns off automatically 5 time units after the last time 'on' was pressed. In [10], the system is given as an MITL formula (with past temporal operators) over atomic propositions $\{\ell, on, off\}$. While we can make use of projections to remove the past temporal operators [28, 44], it turned out that the resulting formula is too large. For this reason, we model the system directly as an SA $\mathcal{A}_{lamp}$ (Figure 4). Then, via Proposition 10 and Theorem 11, we perform the same verification tasks as [10]: **1.** checking the emptiness of $\mathcal{A}_{lamp}$; **2.** model-checking $\mathcal{A}_{lamp}$ against $\varphi_1 = \mathbf{G}_{[0,\infty)}\left(\mathbf{F}_{[0,5]}(\neg\ell)\right)$, i.e. the lamp never stays lit for more than 5 time units; **3.** model-checking $\mathcal{A}_{lamp}$ against $\varphi_2 = \mathbf{F}_{[0,\infty)}\left(\mathbf{G}_{[0,5]}\ell\right) \Rightarrow \mathbf{F}_{[0,\infty)}\left(on \wedge \mathbf{F}_{(0,5]}on\right)$, i.e. if at some point the light stays on for more than 5 time units, then there is an instant when 'on' is pressed, and then it is pressed again before 5 time units. The execution times (with and without minimality criteria) are given in Table 3, where we also reproduce the execution times reported in [10]. Again, these numbers are not meant to be compared directly.

6 Conclusion and future work

We proposed a translation from MITL to signal automata based on the same principles as our previous work in the pointwise setting [15]. The main advantages of this translation over the existing ones are that it is compositional and integrates easily with existing tools. To the best of our knowledge, this is the first practical automata-based approach to MITL model-checking over signals. We plan to add to MightyL support for general MITL operators (either via rewriting or directly by components) and other temporal operators (such as those from ECL [28]). On the theoretical side, a possible future direction is to investigate whether the translation can be generalised (possibly with the techniques in [18] or [42]) to deal with signals that are not necessarily finitely-variable.

Table 3 Execution times for the verification tasks in [10].

Task	Our approach	Our approach w/o minimality	[10]
$\mathcal{S}(\mathcal{A}_{lamp}) = \emptyset$?	1.15s	-	4.24s
$\mathcal{S}(\mathcal{A}_{lamp} \times \mathcal{A}_{\neg\varphi_1}) = \emptyset$?	1.79s	1.59s	17.2s
$\mathcal{S}(\mathcal{A}_{lamp} \times \mathcal{A}_{\neg\varphi_2}) = \emptyset$?	2.53s	196s	257.1s

References

1 Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

4 Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

5 Rajeev Alur and Thomas A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *33rd Annual Symposium on Foundations of Computer Science*, pages 177–186. IEEE, 1992.

6 Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993. doi:10.1006/inco.1993.1025.

7 Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–203, January 1994. doi:10.1145/174644.174651.

8 Eugene Asarin, Paul Caspi, and Oded Maler. A kleene theorem for timed automata. In *LICS'97*, pages 160–171. IEEE Computer Society Press, 1997.

9 Ezio Bartocci, Luca Bortolussi, and Laura Nenzi. A temporal logic approach to modular design of synthetic biological circuits. In *CMSB'13*, volume 8130 of *LNCS*, pages 164–177. Springer, 2013. doi:10.1007/978-3-642-40708-6.

10 Marcello M. Bersani, Matteo Rossi, and Pierluigi San Pietro. A tool for deciding the satisfiability of continuous-time metric temporal logic. *Acta Informatica*, 53(2):171–206, 2016. doi:10.1007/s00236-015-0229-y.

11 Patricia Bouyer, Maximilien Colange, and Nicolas Markey. Symbolic optimal reachability in weighted timed automata. In *CAV'16*, volume 9779 of *LNCS*, pages 513–530. Springer, 2016. doi:10.1007/978-3-319-41528-4.

12 Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. KRONOS: A model-checking tool for real-time systems (tool-presentation). In *FTRTFT'98*, volume 1486 of *LNCS*, pages 298–302. Springer, 1998. doi:10.1007/BFb0055357.

13 Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On MITL and alternating timed automata. In *FORMATS'13*, volume 8053 of *LNCS*, pages 47–61. Springer, 2013.

14 Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On MITL and alternating timed automata of infinite words. In *FORMATS'14*, volume 8711 of *LNCS*. Springer, 2014.

15 Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. MightyL: A compositional translation from MITL to timed automata. In *CAV'17*, LNCS. Springer, 2017. URL: https://hal.archives-ouvertes.fr/hal-01525524.

16 Peter E. Bulychev, Alexandre David, Kim G. Larsen, and Guangyuan Li. Efficient controller synthesis for a fragment of $\mathrm{MTL}_{0,\infty}$. *Acta Informatica*, 51(3-4):165–192, 2014. doi:10.1007/s00236-013-0189-z.

17 Koen Claessen, Niklas Een, and Baruch Sterin. A circuit approach to LTL model checking. In *FMCAD'13*. IEEE, 2013.

18 Julien Cristau. Automata and temporal logic over arbitrary linear time. In *FSTTCS'09*, volume 4 of *LIPIcs*, pages 133–144. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.

19 Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, September 2011. doi:http://dx.doi.org/10.1145/1995376.1995394.

20 Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. Formal requirement debugging for testing and verification of cyber-physical systems. Research Report 1607.02549, arXiv, 2016.

21 Deepak D'Souza and RM Matteplackel. A clock-optimal hierarchical monitoring automaton construction for mitl. Research Report 2013-1, IIS, 2013. URL: `http://www.csa.iisc.ernet.in/TR/2013/1/lics2013-tr.pdf`.

22 Deepak D'Souza, M Raj Mohan, and Pavithra Prabhakar. Eliminating past operators in metric temporal logic. *Perspectives in Concurrency*, pages 86–106, 2008.

23 A. Pnueli E. Asarin, O. Maler and J. Sifakis. Controller synthesis for timed automata. In *SSC'98*, pages 469–474. Elsevier, 1998.

24 Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *CAV'01*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001.

25 Marc Geilen. An improved on-the-fly tableau construction for a real-time temporal logic. In *CAV'03*, volume 2725 of *LNCS*, pages 394–406. Springer, 2003. `doi:10.1007/978-3-540-45069-6_37`.

26 Marc Geilen and Dennis Dams. An on-the-fly tableau construction for a real-time temporal logic. In *FTRTFT*, volume 1926 of *LNCS*, pages 276–290. Springer, 2000. `doi:10.1007/3-540-45352-0_23`.

27 Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV'95*, pages 3–18. Chapman & Hall, 1995.

28 Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *ICALP'98*, volume 1443 of *LNCS*, pages 580–591. Springer, 1998. `doi:10.1007/BFb0055086`.

29 Yoram Hirshfeld and Alexander Moshe Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1):1–28, 2004.

30 Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. LTSmin: High-performance language-independent model checking. In *TACAS'15*, volume 9035 of *LNCS*, pages 692–707. Springer, 2015.

31 Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. Bounded model checking of an MITL fragment for timed automata. In *ACSD'13*, pages 216–225. IEEE Computer Society, 2013.

32 Dileep Raghunath Kini, Shankara Narayanan Krishna, and Paritosh K. Pandya. On construction of safety signal automata for $MITL[\mathcal{U}, \mathcal{S}]$ using temporal projections. In *FORMATS*, volume 6919 of *LNCS*, pages 225–239. Springer, 2011. `doi:10.1007/978-3-642-24310-3_16`.

33 Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

34 Alfons Laarman, Mads Chr. Olesen, Andreas Engelbredt Dalsgaard, Kim Guldstrand Larsen, and Jaco van de Pol. Multi-core emptiness checking of timed Büchi automata using inclusion abstraction. In *CAV'13*, volume 8044 of *LNCS*, pages 968–983. Springer, 2013.

35 Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

36 Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to timed automata. In *FORMATS'06*, volume 4202 of *LNCS*, pages 274–289. Springer, 2006.

37 Dejan Nickovic. *Checking Timed and Hybrid Properties: Theory and Applications. (Vérification de propriétés temporisées et hybrides: théorie et applications).* PhD thesis, Joseph Fourier University, Grenoble, France, 2008.

38 Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *LICS'05*, pages 188–197. IEEE Computer Society Press, 2005.

39 Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007.

40 Amir Pnueli and Aleksandr Zaks. On the merits of temporal testers. In *25 Years of Model Checking*, volume 5000 of *LNCS*, pages 172–195. Springer, 2008. doi:10.1007/978-3-540-69850-0_11.

41 Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. Reactive synthesis from signal temporal logic specifications. In *HSCC'15*, pages 239–248. ACM, 2015.

42 Mark Reynolds. The complexity of the temporal logic with "until" over general linear time. *Journal of Computer and System Sciences*, 66(2):393–426, 2003.

43 Kristin Y. Rozier and Moshe Y. Vardi. A multi-encoding approach for LTL symbolic satisfiability checking. In *FM'11*, volume 6664 of *LNCS*, pages 417–431. Springer, 2011. doi:10.1007/978-3-642-21437-0.

44 Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *FTRTFT'94*, volume 863 of *LNCS*, pages 694–715. Springer, 1994.

A Making signal automata bipartite

Proof of Proposition 2. For $\mathcal{A} = (L, L_0, \alpha, X, \beta, \Delta, \mathcal{F})$, define $\mathcal{A}^{bp} = (L^{bp}, L_0^{bp}, \alpha^{bp}, X^{bp}, \beta^{bp}, \Delta^{bp}, \mathcal{F}^{bp})$ where

- $L^{bp} = \{\ell^s, \dot{\ell}^s, \ell^o \mid \ell \in L\}$;
- $L_0^{bp} = \{\ell^s \mid \ell \in L_0\}$;
- $\alpha^{bp}(\ell^s) = \alpha^{bp}(\dot{\ell}^s) = \alpha^{bp}(\ell^o) = \alpha(\ell)$ for every $\ell \in L$;
- $X^{bp} = X \cup \{y\}$ where y is a fresh clock;
- $\beta^{bp}(\ell^s) = \beta^{bp}(\dot{\ell}^s) = \beta(\ell) \wedge y = 0$, $\beta^{bp}(\ell^o) = \beta(\ell)$ for every $\ell \in L$;
- $\Delta^{bp} = \{(\ell_1^s, \lambda, \ell_2^o), (\dot{\ell}_1^s, \lambda, \ell_2^o), (\ell_1^o, \lambda \cup \{y\}, \ell_2^s) \mid (\ell_1, \lambda, \ell_2) \in \Delta\}$
  $\cup \{(\ell^o, \{y\}, \dot{\ell}^s), (\ell^s, \emptyset, \ell^o) \mid \ell \in L\}$;
- $\mathcal{F}^{bp} = \{\{\ell^s, \ell^o \mid \ell \in F\} \mid F \in \mathcal{F}\}$.

Intuitively, we create three copies ℓ^s, $\dot{\ell}^s$, ℓ^o of each location ℓ of $\mathcal{A}$ and use the clock y to enforce the desired behaviour. In particular, the 'dotted' locations $\dot{\ell}^s$ are used to deal with the situation where the 'source' interval is right-closed. One can verify that $\mathcal{A}^{bp}$ is bipartite (let $L^s = \{\ell^s, \dot{\ell}^s \mid \ell \in L\}$) and $\mathcal{S}(\mathcal{A}) = \mathcal{S}(\mathcal{A}^{bp})$. ◀

B Signal components for the 'release' operators

The component $\mathcal{C}_\chi$ for $\chi = \psi_1 \mathcal{R} \psi_2$ (Figure 5) is based on similar ideas as the component for $\psi_1 \mathcal{U} \psi_2$. In this case, an obligation can be satisfied by either $\widehat{\psi_1} \wedge \widehat{\psi_2}$ holding in a singular interval or $\widehat{\psi_1} \wedge * \psi_2$ holding in an open interval.

The component $\mathcal{C}_\chi$ for $\chi = \psi_1 \mathcal{R}_{(0,a)} \psi_2$ is given in Figure 6. In this case, all old obligations are implied by the newest one. We therefore reset the clock x when p_χ becomes false. The component for $\psi_1 \mathcal{R}_{(0,a]} \psi_2$ is similar and hence omitted.

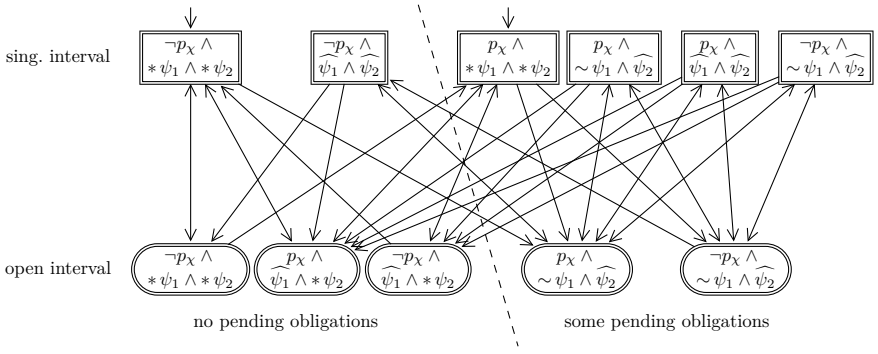

Figure 5 The component SA $\mathcal{C}_\chi$ for $\chi = \psi_1 \mathcal{R} \psi_2$.

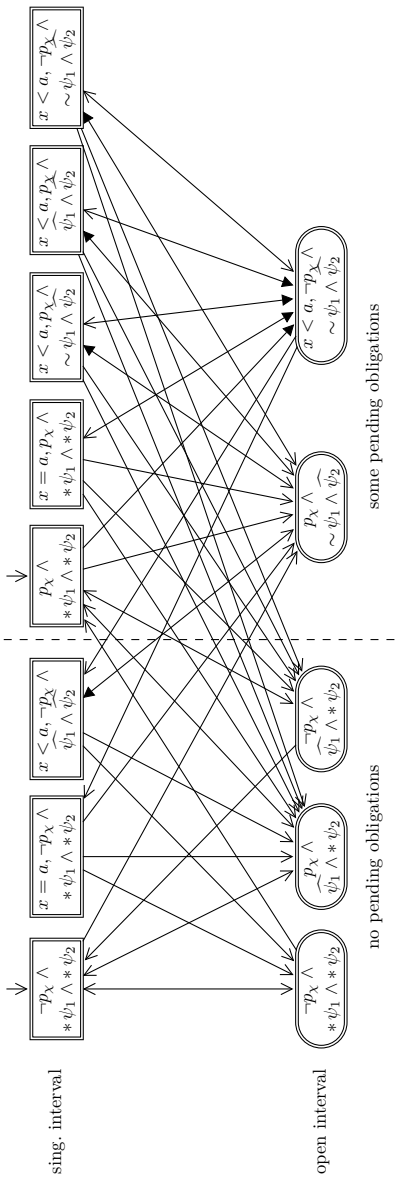

Figure 6 The component SA $\mathcal{C}_\chi$ for $\chi = \psi_1 \,\mathcal{R}_{(0,a)}\, \psi_2$. The transitions with ▶ reset x.