Timed-Automata-Based Verification of MITL over Signals

Thomas Brihaye¹, Gilles Geeraerts², Hsi-Ming Ho³, and Benjamin Monmege⁴

¹ Université de Mons, Mons, Belgium
thomas.brihaye@umons.ac.be
² Université libre de Bruxelles, Brussels, Belgium
gigeerae@ulb.ac.be
³ Université de Mons, Mons, Belgium
hsi-ming.ho@umons.ac.be
⁴ Aix Marseille Univ, CNRS, LIF, Marseille, 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 \( \omega \)-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.2017.7

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, computers interact with an environment that is intrinsically continuous, and ensuring complex 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 [11], TiAMO [10]...As far as logics are concerned, several proposals have been made in the literature during the past 30 years (such

* This work has been supported by the FRS/F.N.R.S. PDR grant SyVeRLo, and (partially) funded by the DeLTA project (ANR-16-CE40-0007) and the SensAS project (INS2I JCJC’17).
7:2  Timed-Automata-Based Verification of MITL over Signals

as MTL [33], TPTL [6], 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 [3] for more practical motivations [36, 12, 31, 13, 15, 9].

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’ [3]. 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 [5, 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 [3, 36, 9]). 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 state 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 [7, 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 [8], etc. Apart from these practical considerations, the difference between these two semantics matters as it changes the expressive power of the logic. For example, the following formula (which requires $p$ to hold exactly in $[0,a]$ for some $a \geq 0$) is satisfiable in the continuous semantics only: $p \land F(\neg p) \land G(\neg p \Rightarrow G(\neg p)) \land \neg (p U (\neg p))$.

Contribution. In order to remedy the lack of comprehensive tool support for MITL in the pointwise semantics, we have recently introduced MIGHTYL [14], 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 satisfiability 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 $\varphi$ into a network of signal automata $C_{init} \times \prod \chi C_\chi$, one for each subformula $\chi$ in $\varphi$, plus an extra signal automaton $C_{init}$ (extending the ideas of our previous work [14] 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).

---

1 As for MTL, for instance, which becomes decidable on finite words in the pointwise semantics [38].
Concretely, given a signal automaton \( A \) modelling a system, and a property \( \varphi \) to be checked on \( A \), we can perform model-checking by: (i) building the network of signal automata \( C_{\text{init}} \times \prod_{\chi} C_{\chi} \) from \( \neg \varphi \) using the procedure of Section 3; (ii) translating, using the techniques of Section 4, \( A \), \( C_{\text{init}} \) and all \( C_{\chi} \) into corresponding timed automata \( B^A \), \( B_{\text{init}} \) and \( B_{\chi} \) (for all subformulae \( \chi \)) respectively; and (iii) checking (using a model-checker for timed automata) whether the language \( B^A \times B_{\text{init}} \times \prod_{\chi} B_{\chi} \) is empty. If this is the case, then our translation ensures that the language of \( A \times C_{\text{init}} \times \prod_{\chi} C_{\chi} \) is empty, which in turn holds if and only if \( A \models \varphi \), by construction. We have implemented this approach as 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 [4] to signal automata. The translation works by rewriting the input formula into one with only past operators using projections [21]. 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 \( GF p \) 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 [36, 37, 20]. 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 [40, 16, 23]; that is, suppose we introduce a new proposition \( p_{\chi} \) for the subformula \( \chi = \varphi_1 U \varphi_2 \), a tester enforces \( p_{\chi} \Leftrightarrow \varphi_1 U \varphi_2 \) to hold at all times while a positive tester only enforces the weaker formula \( p_{\chi} \Rightarrow \varphi_1 U \varphi_2 \) 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 [3] 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 [26, 25] 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 [18] to satisfiability/model-checking for MITL over signals [31, 9]. The SMT approach is straightforward to implement and there are publicly available tools.

\(^2\) These models are, however, not more expressive than signal automata.
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.

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 $\sigma$ over $AP$ is a subset of $2^{AP}$. A signal $\gamma$ 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 $\sigma = \sigma_0\sigma_1\sigma_2\cdots$ over $2^{AP}$ is an infinite sequence of states $\sigma_i \in 2^{AP}$. An interval sequence $I = I_0I_1I_2\cdots$ 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 to be 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 $\kappa = (\sigma, I)$ where $\sigma$ is a state sequence over $2^{AP}$ and $I$ is an interval sequence. 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) $\kappa$ 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:

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

where $p \in AP$ 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 $\gamma$ over $2^{AP}$, $t \in \mathbb{R}_{\geq 0}$, and an MITL formula $\varphi$, the satisfaction relation $\gamma, t \models \varphi$ is defined as follows (following [3], we adopt the strict-future semantics for the temporal operators):

- $\gamma, t \models \top$; $\gamma, t \models p$ if $p \in \gamma(t)$;
- $\gamma, t \models \varphi_1 \land \varphi_2$ 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 $S(\varphi)$ for the set of all signals $\gamma$ such that $\gamma \models \varphi$.

We will use standard syntactic sugar, e.g. $\varphi_1 \lor \varphi_2 \equiv \neg (\neg \varphi_1 \land \neg \varphi_2)$, $\bot \equiv \neg \top$, $\varphi_1 \Rightarrow \varphi_2 \equiv \neg \varphi_1 \lor \varphi_2$, the ‘eventually’ operator $\mathcal{F}_I \varphi \equiv \top \mathcal{U}_I \varphi$, the ‘globally’ operator $\mathcal{G}_I \varphi \equiv \neg \mathcal{F}_I \neg \varphi$, 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 \( \varphi \) 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 classical timed automata afterwards. Like \cite{3}, we equip these automata with generalised Büchi acceptance conditions. From now on, a propositional constraint \( \phi \) over \( AP \) is a set of states over \( AP \); that we denote by means of a Boolean formula over \( AP \). For example, assuming \( AP = \{p, q, r\} \), the propositional constraint \( p \land \neg q \) denotes \( \{\{p, r\}, \{p\}\} \). Let \( X \) be a finite set of clocks. The set \( G(X) \) of clock constraints \( g \) over \( X \) is generated by the grammar \( g := T \mid \bot \mid g \land g \mid x \bowtie c \) where \( \bowtie \in \{\leq, <, \geq, >\} \), \( x \in X \) and \( c \in \mathbb{N} \). A valuation \( v \) of \( X \) is a mapping \( v : X \rightarrow \mathbb{R}_{\geq 0} \).

We denote by \( 0 \) 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 denoted \( v \models g \). 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 \( A = (L, L_0, \alpha, X, \beta, \Delta, F) \) where

- \( L \) is a finite set of locations;
- \( L_0 \subseteq L \) is the set of initial locations;
- \( \alpha \) is the location labelling function that assigns to each location \( \ell \in L \) a propositional constraint \( \alpha(\ell) \subseteq 2^{AP} \);
- \( X \) is a finite set of clocks;
- \( \beta \) is the location labelling function that assigns to each location \( \ell \in L \) a clock constraint \( \beta(\ell) \in G(X) \);
- \( \Delta \subseteq L \times 2^X \times L \) 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;
- \( F \subseteq 2^L \) is the family of sets of accepting locations.

A run \( \pi \) of \( A \) on a signal \( \gamma \) over \( 2^{AP} \) is an infinite sequence of the following form:

\[
(\ell_0, I_0) \xrightarrow{v_1} (\ell_1, I_1) \xrightarrow{\lambda_1} (\ell_2, I_2) \xrightarrow{\lambda_2} \cdots
\]

where:

- (1) for all \( i \geq 0 \), \( \ell_i \) is a location of \( A \); (2) the sequence \( I_0I_1I_2 \cdots \) 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:

- **Initialisation:** \( \ell_0 \in L_0 \) and \( v_0 = 0 \); and
- **Consequence:** 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:** \( v_\pi(t) = \beta(\ell_\pi(t)) \) for all \( t \geq 0 \), assuming \( v_\pi(t) = v_i + (t - \text{inf}(I_i)) \) and \( \ell_\pi(t) = \ell_i \) if \( t \in I_i \) for some \( i \geq 0 \); and
- **Adequacy:** \( \gamma(t) = \alpha(\ell_\pi(t)) \) for all \( t \geq 0 \).

We say that \( \pi \) is bipartite if \( I_0I_1I_2 \cdots \) is bipartite. We say that \( \pi \) is accepting if for all \( F \in F : \{i \mid \ell_i \in F\} \) is infinite. A signal \( \gamma \) is accepted by \( A \) if there is an accepting run of \( A \) on \( \gamma \).

We write \( S(A) \) for the set of signals accepted by \( A \). For two SAs \( A_1 \) and \( A_2 \), we denote by \( A_1 \times A_2 \) their (asynchronous) product, defined in a manner similar to \cite{3}: intuitively, in each location of this product, we can either fire only a transition of \( A_1 \) (provided that the guard in the current location of \( A_2 \) holds after the transition), or only a transition of \( A_2 \), or one in \( A_1 \) and one in \( A_2 \), provided that the guards on their (respective) target locations are satisfied afterwards. In particular, we have \( S(A_1 \times A_2) = S(A_1) \cap S(A_2) \).
We focus on the class of bipartite SA whose runs are bipartite by construction. An SA $A = (L, L_0, \alpha, X, \beta, \Delta, F)$ is bipartite if there exists a partition of $L$ into $L^s$, $L^o$ respecting the conditions given hereinafter. Intuitively, on reading a signal $\gamma$, $A$ is in a location of $L^s$ ($L^o$) when it sees a singular (respectively open) interval of $\kappa \in \llbracket \gamma \rrbracket_{bp}$:

- $L_0 \subseteq L^s$
- if $(\ell_1, \lambda, \ell_2) \in \Delta$ then $\ell_1 \in L^s$ if and only if $\ell_2 \in L^o$;
- 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 $\ell_1 \in L^o$ (and thus $\ell_2 \in L^s$), 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 from [3] (see Appendix A for a proof):

**Proposition 2.** Any SA $A$ can be turned into a bipartite SA $A_{bp}$ such that $S(A) = S(A_{bp})$.

From now on, when depicting bipartite SA, we use rectangle and rounded rectangles for the locations from $L^s$ and $L^o$ 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 $\varphi$, whether $S(\varphi) \neq \emptyset$ (if it is the case, we say that $\varphi$ is satisfiable). The model-checking problem asks, given an SA $A$ and an MITL formula $\varphi$ whether $S(A) \subseteq S(\varphi)$. If it is the case, we write $A \models \varphi$.

## 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 [14]: we keep track of the satisfiability of each temporal subformula (i.e. a subformula whose outermost operator is temporal) $\chi$ with an SA $C_\chi$. From now on, we fix a set $AP$ of atomic propositions and a negative normal form MITL formula $\varphi$ 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]. Moreover, we regard all temporal subformulae of $\varphi$ as distinct formulae.

**Triggers.** Let $\Phi$ be the set of temporal subformulae of $\varphi$. We introduce a new atomic proposition $p_\chi$ for each $\chi \in \Phi$ and we let $AP_\varphi = \{p_\chi \mid \chi \in \Phi\}$. Each $p_\chi$ is called a trigger (for $\chi$). Intuitively, pulling the trigger $p_\chi$ (i.e. setting $p_\chi$ to true) at some point means that $\chi$ is required to hold at that point. On the other hand, $p_\chi$ being false at some point does not mean that $\chi$ 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 $\chi$ is a subformula of $\psi$, the component SA $C_\psi$ will pull the trigger $p_\chi$ whenever the satisfaction of $\chi$ is needed to check the value of $\psi$. 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

\(^3\) Note that a product of bipartite SAs is a bipartite SA.
is the point of the formulae $\overline{\psi}$, $*\psi$, $\sim\psi$ and $\hat{\psi}$ that we introduce hereinafter. Concretely, the outcome of our construction for an MITL formula $\varphi$ is a network of SA that accepts an AP$_\varphi$-decorated version of $S(\varphi)$. In other words, the signals accepted by our construction are over $AP \cup AP_\varphi$ and their projections on AP yields $S(\varphi)$, as stated in Theorem 8 at the end of the section.

For each (not necessarily temporal) subformula $\psi$ of $\varphi$, we denote by $P_\psi$ the set of atomic propositions $p_\chi \in AP_\varphi$ such that $\chi$ is a top-level temporal subformula of $\psi$, i.e. the outermost operator of $\chi$ is $U_I$ or $R_I$, yet $\chi$ does not occur under the scope of another $U_I$ or $R_I$ in $\psi$. For instance, $P_{\psi_1 \lor \psi_2 \lor (U_I (R_I \psi_3) \lor (U_I \psi_4) \lor (\psi_5 \lor (U_I \psi_5)))} = \{p_\psi_1, p_\psi_2, p_\psi_3, p_\psi_4, p_\psi_5\}$. For a signal $\gamma'$ over $2^P$ (where $P'$ is a set of atomic propositions) and $P \subseteq P'$, we denote by $proj_P(\gamma')$ the projection of $\gamma'$ onto $P$, i.e. the signal obtained from $\gamma'$ by hiding all the atomic propositions $p \notin P$. For a set of signals $S$ over $2^P$ and $P \subseteq P'$, we write $proj_P(S) = \{proj_P(\gamma') \mid \gamma' \in S\}$. Conversely, we say a signal $\gamma'$ over $2^P$ extends a signal $\gamma$ over $2^P (P \subseteq P')$ if $proj_P(\gamma') = \gamma$.

**Formulae over $AP \cup AP_\varphi$.** We define some syntactic operations on Boolean combinations over $AP \cup AP_\varphi$ that will be used in the components described later. Specifically, for a subformula $\psi$ of $\varphi$, we define formulae $\overline{\psi}$ (introducing the trigger variables), $*\psi$ (ensuring that we do not pull any trigger of $\psi$), $\sim\psi$ (checking that $\psi$ does not hold, while none of its triggers are pulled), and $\hat{\psi}$ (checking $\psi$ while triggering a minimal set of triggers).

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

\[
\begin{align*}
\overline{\psi_1 \land \psi_2} &= \overline{\psi_1} \land \overline{\psi_2} & \overline{\psi} &= \psi \text{ when } \psi \text{ is } T \text{ or } \perp \text{ or } p \text{ or } \neg p \\
\overline{\psi_1 \lor \psi_2} &= \overline{\psi_1} \lor \overline{\psi_2} & \overline{\psi} &= p_\psi \text{ when } \psi \text{ is } \psi_1 U_I \psi_2 \text{ or } \psi_1 R_I \psi_2.
\end{align*}
\]

The formula $*\psi$, read as “do not pull the triggers of $\psi$”, is used to ensure that our components only follow the ‘minimal models’ of $\psi$. It is defined as the conjunction of the negations of all $p_\chi \in P_\psi$ (if $P_\psi = \emptyset$ then $*\psi = T$). As a concrete example,

\[
*((\neg p \lor \psi_1 U_I \psi_2) \land (q \lor \psi_3 R_I (\psi_4 U_I \psi_5))) = \neg p_\psi_1 U_I \psi_2 \land \neg p_\psi_3 R_I (\psi_4 U_I \psi_5).
\]

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

\[
\begin{align*}
mm(T) &= T & mm(\perp) &= \perp & mm(p) &= p & mm(\neg p) &= \neg p \\
mm(\phi_1 \land \phi_2) &= (mm(\phi_1) \land \sim \phi_2) \lor (mm(\phi_2) \land \sim \phi_1) \lor ((mm(\phi_1) \land \phi_2) \land (mm(\phi_1) \land \phi_2)) \\
mm(\phi_1 \lor \phi_2) &= mm(\phi_1) \lor mm(\phi_2)
\end{align*}
\]

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

\[\textbf{Proposition 3. For a subformula } \psi \text{ of } \varphi, \text{ if } \sigma \models \overline{\psi} \text{ for some state } \sigma \text{ over } AP \cup P_\psi, \text{ there is a state } \sigma' \text{ over } AP \cup P_\psi \text{ such that } \sigma' \models \hat{\psi} \text{ and } proj_{AP}(\sigma) = proj_{AP}(\sigma') \text{ (and vice versa).}\]

\[\textbf{Proof. By induction on the structure of } \overline{\psi}. \text{ For the direct implication, if } \overline{\psi} = \overline{\psi_1} \lor \overline{\psi_2} \text{ then one of the following must hold:}\]

1. $\sigma \models \overline{\psi_1}$ and $\sigma \not\models \overline{\psi_2}$: apply the induction hypothesis on $\sigma \setminus P_{\psi_2}$ and $\overline{\psi_1}$ (note that $P_{\psi_1} \cap P_{\psi_2} = \emptyset$, and $\psi_2$ is in negative normal form).

\[4 \text{ Note that the size of } \hat{\psi} \text{ is at most quadratic in the size of } \psi.\]
Minimal of triggers. The real impact of $\hat{\psi}$ with respect to $\psi$ is to ensure the minimality of triggers pulled during an execution. Indeed, now show that if $\varphi$ is satisfied by a signal $\gamma$ (over $2AP$), then there must be a way to extend $\gamma$ into a signal $\gamma'$ over $2AP\cup AP_\varphi$ such that the triggers $AP_\varphi$ are only pulled when necessary in $\gamma'$, 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 $\varphi$. This observation is formalised in the following two propositions.

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

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

**Proposition 5.** For a signal $\gamma$ over $2AP\cup \{p_\chi\}$ where $\varphi \in \Phi$ and either $\chi = \psi_1 U_I \psi_2$ or $\chi = \psi_1 R_I \psi_2$, 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 $\gamma'$ over $2AP\cup \{p_\chi\} \cup P_\psi \cup P_\varphi$ extending $\gamma$ such that

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

  \[ \text{Expand}_\chi = \left[ \left( \psi_1 \wedge \neg \psi_2 \right) U_I \left( * \psi_1 \wedge \psi_2 \right) \right] \vee \left[ \left( \psi_1 \wedge \psi_2 \right) U T \right] \]

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

  \[ \text{Expand}_\chi = \left[ \left( \sim \psi_1 \wedge \psi_2 \right) U_I \left( * \psi_1 \wedge \psi_2 \right) \right] \vee \left[ \left( \psi_1 \wedge * \psi_2 \right) U T \right] \vee \left[ G_I \left( \sim \psi_1 \wedge \psi_2 \right) \right] \]

- for each $p_\theta \in P_\psi \cup P_\varphi$, we have $\text{proj}_{AP\cup \{p_\theta\}}(\gamma')$, $t \models (p_\theta \Rightarrow \theta)$ for all $t \in \mathbb{R}_{\geq 0}$.

**Proof.** Assume that $\chi = \psi_1 U_I \psi_2$ and let $\zeta$ be a signal over $2AP\cup \{p_\chi\} \cup P_\psi \cup P_\varphi$ extending $\gamma$ such that $p_\theta \in \zeta(t)$ if and only if $\gamma, t \models \theta$ for each $p_\theta \in P_\psi \cup P_\varphi$ 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 $\text{inf}(f) = 0$):

- 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 $\gamma$, $\psi_1 \wedge \neg \psi_2$ 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 $\zeta$ via Proposition 3. More precisely, we apply Proposition 3 to constant segments of $\zeta$ and $\psi_1$, $\psi_2$, or both $\psi_1$ and $\psi_2$, as required by the interpretation of $p_\chi$ in $\gamma$. 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 $\chi = \psi_1 \mathsf{R} \psi_2$. The other implication is simpler.

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

\textbf{The components.} We are now ready to present the components $C_{\chi}$ for $\chi \in \Phi$.

The component $C_{\chi}$ for $\chi = \psi_1 \mathsf{U} \psi_2$ is given in Figure 1. We now explain how it has been produced. Thanks to Proposition 2, we provide a bipartite $\mathsf{SA}$ (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 $\ell_0^0$ and $\ell_0^o$, that are used as long as trigger $p_\chi$ is not pulled: then, there is no need to pull any trigger of $\psi_1$ nor $\psi_2$, which is ensured via the use of formula $\star \psi_1 \wedge \star \psi_2$. Consider then the first time when trigger $p_\chi$ is pulled (by another component automaton): it is either in a singular interval in which case we jump into location $\ell_1^0$ (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 $\psi_1$ or $\psi_2$, thus checking $\star \psi_1 \wedge \star \psi_2$), or in an open interval in which case we jump either into location $\ell_1^1$ if $\psi_2$ does not hold (i.e. if $\sim \psi_2$ holds), or into location $\ell_1^3$ if $\psi_2$ holds (i.e. if $\overline{\psi_2}$ is in the guard) which fulfills right away the new obligation (notice that, in the figure, we did not put $p_\chi$ in the guard of this location, for simplification: we will discuss this point more in detail afterwards).

When $p_\chi$ is first pulled in an open interval (which means we jump into location $\ell_1^1$ or $\ell_1^3$), by the semantics of the ‘until’ operator, $\psi_1$ must also hold in that interval. When in $\ell_1^3$, the successors are the same as in $\ell_0^3$. When in $\ell_1^1$ with a pending obligation, there are two cases for the next jump:

- either $\psi_2$ holds in the next singular interval, and then no trigger of $\psi_1$ needs to be pulled (i.e. guard $\star \psi_1 \wedge \overline{\psi_2}$): if there are no new pulled trigger $p_\chi$, we jump into location $\ell_3^3$; otherwise, we jump into location $\ell_4^3$ 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 $\psi_2$ does not hold, in which case $\psi_1$ should hold (i.e. guard $\bar{\psi}_1 \wedge \sim \psi_2$): we then jump into location $\ell_2^2$ whether or not a new trigger $p_\chi$ is pulled.

When $p_\chi$ is first pulled in a singular interval (which means we jump into location $\ell_1^0$), there is no need to pull any trigger of $\psi_1$ nor $\psi_2$. Then, while in one of the ‘singular’ locations $\ell_1^0$, $\ell_2^2$ or $\ell_4^3$, with a pending obligation, in the next jump, there are two cases:
either $\psi_2$ holds in the next open interval, in which case $\psi_1$ should still hold (because of the semantics of the `until' operator): we can jump into the previously introduced location $\ell_2^3$.

- or $\psi_2$ does not hold (then, $\psi_1$ should hold anyway) and we jump either in location $\ell_2^1$ if a new trigger $p_x$ is pulled, or in $\ell_2^2$ is no new trigger $p_x$ is pulled. Location $\ell_2^2$ has the same successors as $\ell_2^1$ but we still need to distinguish them since $\ell_2^1$ must check that a new pending obligation is pulled.

Initially, we do not want to pull any trigger of $\psi_1$ or $\psi_2$, therefore, $\ell_0^3$ and $\ell_1^1$ are the two initial locations, depending on whether trigger $p_x$ 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 $\ell_0^1$).

Notice that, thanks to the use of $*\psi_1$ and $\tilde{\psi}_1$ formulae, only the necessary triggers in $P_{\psi_1} \cup P_{\psi_2}$ are pulled during an execution of this component. Indeed, this is not true for location $\ell_3^3$: when going from locations $\ell_0^3$ or $\ell_3^3$, to pull only minimal sets of triggers, we must make sure to go in $\ell_3^3$ only when a new trigger $p_x$ is pulled. This requires to split this location into two (one where $p_x$ 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 $C_\chi$ is the one for $\chi = \psi_1 \cup_{(0,a)} \psi_2$ (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 $\ell_1^1$, $\ell_2^1$, and $\ell_3^1$ from locations $\{\ell_0^1, \ell_2^3, \ell_0^3, \ell_2^1, \ell_3^1\}$. Now, the valuation of $x$ would fix a deadline for the satisfaction of $\psi_2$. Indeed, as long as $\psi_2$ does not hold, we must check that $x < a$. When $\psi_2$ 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 $\psi_2$ 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 set si to false. Then, the clock constraint that must be checked in singular interval (whether or not ψ2 is currently fulfilled) is not x < a but g defined by (si ∧ x < a) ∨ (¬si ∧ x ≤ a): in particular, the guard g in location ℓ3 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 fulfill ψ2 at time x = a, but then, the only fireable transitions are the one towards ℓ3 and ℓ3, where ψ2 then holds. This also explains why guard g does not need to be checked when entering in ℓ3 or ℓ3.

Second, this cannot be done as such when entering location ℓ1 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 ℓ1 or ℓ4.

The component for ψ1 U_{0,a} ψ2 is similar and hence omitted. The components for ‘release’ operators follow the same pattern as the ones for ‘until’. Due to lack of space, we present them in Appendix B. Then:

▶ Proposition 7. For each χ ∈ Φ, the component Cχ accepts exactly all signals γ over 2AP∪APs such that γ, t |− pχ ⇒ Expandχ for all t ∈ R≥0 (where Expandχ is one of the formulae in Proposition 5).

Finally, we need a simple initial component Cinit which enforces ̃ϕ at t = 0 and *ϕ at all t > 0, as suggested by Proposition 5. We can now state the main theorem of this section.

▶ Theorem 8. projAP(S(Cinit × Πχ∈Φ Cχ)) = S(ϕ).

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 A into a timed automaton B_A that accepts exactly the ‘timed words’ counterparts of the signals accepted by A. Moreover, the construction can be used in a compositional manner: if A is the product of a number of components SAs, B_A can be obtained as the product of the TA that result from applying the construction to the components of 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 the formal definition of timed automata.

Timed words and timed automata. A time sequence is an infinite sequence σ = τ0τ1τ2... of non-negative reals (called timestamps) such that (1) τ0 = 0; (2) for all i ≥ 0, τi ≤ τi+1; (3) for all t ∈ R≥0, there is some i ≥ 0 such that τi > t. A timed word ρ = (σ, σ) over 2AP is a pair of a state sequence σ over 2AP and a time sequence τ. Alternatively, we may see ρ as an infinite sequence (σ0, τ0)(σ1, τ1)(σ2, τ2)... of events (σ, τ). We now define timed automata, with generalised acceptence conditions as before (used by [27] in the untimed setting).

▶ Definition 9. A timed automaton (TA) over 2AP is a tuple A = (L, L0, X, Δ, F) where
- L is a finite set of locations;
- L0 ⊆ L is the set of initial locations;
- X is a finite set of clocks;
A run π of A on a timed word ρ = (σ₀, τ₀)(σ₁, τ₁)(σ₂, τ₂) · · · over 2ⁿ is an infinite sequence

\[(\ell₀, v₀) \xrightarrow{λ₁, σ₀, d₀} (\ell₁, v₁) \xrightarrow{λ₂, σ₁, d₁} (\ell₂, v₂) \xrightarrow{λ₃, σ₂, d₂} \cdots \]

where, for all i ≥ 0: (1) ℓᵢ is a location of A; (2) vᵢ is a valuation of X; (3) dᵢ = τᵢ - τᵢ₋₁ (assuming τ₋₁ = 0). WeMoreover, for all \( \ell \) of \( I \) and that satisfies the following:

- **Initial**: \( \ell₀ \in L₀ \); and
- **Conclusion**: for all i ≥ 0: (\( \ellᵢ, φ, g, ℓᵢ₊₁, ℓᵢ₊₂) ∈ Δ with \( σᵢ ∈ φ \) and \( vᵢ + dᵢ \models g \); and
- **Timing**: for all i ≥ 0, \( vᵢ₊₁ = (vᵢ + dᵢ)[λᵢ₊₁ ← 0] \).

We say that \( π \) is **accepting** if for all accepting sets \( F \in F \), the set \{i | ℓᵢ ∈ F\} is infinite. A timed word \( ρ \) is **accepted** by \( A \) if there is an accepting run of \( A \) on \( ρ \). We write \( L(A) \) for the set of timed words accepted by \( A \). For two TAs \( A₁ \) and \( A₂ \), we denote by \( A₁ × A₂ \) their (synchronous) product [2]. In particular, we have \( L(A₁ × A₂) = L(A₁) ∩ L(A₂) \).

**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 \( κ = (σ₀, I₀)(σ₁, I₁) \cdots \) s.t. \( κ ∈ [γ]^{∞} \) (i.e., \( Iᵢ \) is singular for all even \( i ≥ 0 \)), we define:

\[ [κ]_{tw} = (σ₀, sup(I₀))(σ₁, inf(I₁))(σ₂, sup(I₂))(σ₃, inf(I₃)) \cdots. \]

Note that we represent a state change at time \( t \) by two events with timestamp \( t \) (note that \( sup(Iᵢ) = inf(Iᵢ₊₁) \) for each even \( i ≥ 0 \)). Abusing notations, we write \([γ]_{tw} = \{[κ]_{tw} | κ ∈ [γ]^{∞}\} \) and \([S]_{tw} = \bigcup_{γ ∈ S} [γ]_{tw} \) for a set \( S \) of signals.

**Proposition 10.** Given a (bipartite) SA \( A \), we can construct a TA \( B^A \) such that \( L(B^A) = [S(A)]_{tw} \). In particular, if \( A = A₁ × \cdots × Aₙ \) then \( L(B^{A₁} × \cdots × B^{Aₙ}) = [S(A)]_{tw} \).

**Proof (Sketch).** For a clock constraint \( g ∈ G(X) \), let \( g^− \) be the clock constraint obtained from \( g \) by replacing all clauses of the form ‘\( x ≤ c \)’ with ‘\( x < c \)’ and all ‘\( x > c \)’ with ‘\( x ≥ c \)’. Likewise, let \( g^+ \) be the clock constraint obtained from \( g \) by replacing all ‘\( x < c \)’ with ‘\( x ≤ c \)’ and all ‘\( x ≥ c \)’ with ‘\( x > c \)’. The following statements hold (for a valuation \( v \) of \( X \)):

- \( v \models g^− \) if and only if for some \( δ ∈ R_{>0} \), we have \( v + t \models g \) for all \( t ∈ (0, δ] \).
- \( v \models g^+ \) if and only if for some \( δ ∈ R_{>0} \), we have \( v' \models g \) for all valuations \( v' \) of \( X \) such that \( v' + t = v \) for some \( t ∈ (0, δ] \).

In what follows, we write \( g[λ ← 0] \) for the clock constraint obtained from \( g \) by replacing all occurrences of clocks \( x ∈ λ \) with \( 0 \). For \( A = (L, L₀, α, X, β, Δ, F) \) (which by assumption is bipartite and \( L = L^A \cup L^o \)), define \( B = (L^A, L^o, X^A, Δ^A, F^A) \) where

\( L^A = \{\ell | \ell ∈ L^A \} ∪ \{\ell', ε | \ell ∈ L^o \} ∪ \{\ell'| \ell ∈ L^o \} \);  
\( L^o = \{\ell'| \ell ∈ L^o \}; \)
\( X^A = X \cup \{y\} \) where \( y \) is a fresh clock;  
\( Δ^A = \{\{\ell'| \ell ∈ L^A \} ∪ \{\ell'| \ell ∈ L^o \} \} \);  
\( F^A = \{\ell | \ell ∈ L^A \} ∪ \{\ell | \ell ∈ L^o \} \} \mid F ∈ F \).
The clock constraint combinations over \( \mathcal{A} \). As an example, the component \( \mathcal{TA} \) specifies whether the current \( p_k \)-interval is left-closed. The transitions with \( \triangleright \) (respectively, \( \trianglerighttright ) set \( s_i \) to true (respectively, false). The clock constraint \( g \) is defined as \( (s_i \land x < a) \lor (\neg s_i \land x \leq a) \).

Intuitively, the ‘dotted’ locations \( \hat{\ell}^s, \hat{\ell}^o \) are used to allow interleaving and stuttering as \( \mathcal{A} \) stays in \( \ell \in L^x \): this is crucial to make the asynchronous product \( A_1 \times \cdots \times A_n \) and the synchronous product \( B^A_1 \times \cdots \times B^A_n \) match. Finally, for pragmatic reasons, we make suitable modifications to \( B \) to obtain a \textit{strongly non-Zeno} \( \mathcal{TA} B^A \) (i.e. a \( \mathcal{TA} \) in which time progresses), as in [22].

The proposition above works for any (bipartite) \( \mathcal{SA} \). For \( C_{\text{init}} \) or each component \( C_\chi \)(\( \chi \in \Phi \)) in the previous section, however, we can suppress all the ‘dotted’ locations \( \hat{\ell}^s, \hat{\ell}^o \) and build a much simpler \( \mathcal{TA} \) (which we denote by \( \mathcal{B}_{\text{init}} \) or \( \mathcal{B}_\chi \), respectively).

Our main result can then be stated as the following theorem, where the projection operator \( \text{proj} \) is defined in a similar way as in the setting of signals.

\[ \text{Theorem 11. } \text{proj}_{\mathcal{AP}} \left( \mathcal{L}(B^A \times B_{\text{init}} \times \prod_{\chi \in \Phi} B_\chi) \right) = \text{proj}_{\mathcal{AP}} \left( \left[ \mathcal{S}(A \times C_{\text{init}} \times \prod_{\chi \in \Phi} C_\chi) \right]_{tw} \right) \]

for any given \( \mathcal{SA} \mathcal{A} \) over \( 2\text{AP} \cup \text{AP}^o \) whose propositional constraints can be written as Boolean combinations over \( \text{AP} \) (i.e. do not involve atomic propositions in \( \text{AP}^o \)).

As an example, the component \( \mathcal{TA} B_\chi \) for \( \chi = \psi_1 \cup (0,a) \psi_2 \) (in which we use a new atomic proposition \( p_{\text{sing}} \) 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 \textsc{MightyL} [14]. Given a formula \( \varphi \) over \( \text{AP} \) in MITL, the tool generates the model \( \mathcal{TA} B^A \) where \( \mathcal{A} \) is a universal \( \mathcal{SA} \) over \( 2\text{AP} \cup \text{AP}^o \), the initial component \( \mathcal{B}_{\text{init}} \), and the corresponding component \( \mathcal{TA} B_\chi \) for each temporal subformula \( \chi \) of \( \varphi \) in the \textsc{Uppaal} xml format. The user can, of course, replace \( B^A \) with the model \( \mathcal{TMA} \) of their choice and perform model-checking with existing \( \mathcal{TA} \)-based tools.\[^{\text{7}}\]

\[^{\text{7}}\] Our implementation is publicly available and can be executed directly on the webpage: \text{http://www.ulb.ac.be/di/verif/mightyl}. In the following experiments,

\[^{\text{5}}\] Alternatively, one may translate \( \neg\varphi \) into a ‘pointwise’ formula and invoke the approach in [14] directly; the number of temporal subformulae (and hence the number of clocks) would be doubled, however.

\[^{\text{6}}\] More precisely, our tool accepts all temporal operators that are labelled with intervals of the form \((0,\infty), [0,\infty), (0,a), [0,a), (0,a) \text{ or } [0,a)\). If 0 is included in the interval, the temporal operator is given a weak-future interpretation [39], e.g., \( \psi_1 \cup_{[0,a)} \psi_2 \iff \psi_2 \lor (\psi_1 \cup_{[0,a)} \psi_2) \). Remember that general MITL formulae can be rewritten into formulae of this fragment, e.g., \( F_{[a,\infty)} \psi \iff G_{[0,a)} F\psi \).

\[^{\text{7}}\] We require \( M \) to be strongly non-Zeno and \( \mathcal{L}(M) = [\mathcal{S}(A)]_{tw} \) where \( \mathcal{A} \) is an \( \mathcal{SA} \) over \( 2\text{AP} \cup \text{AP}^o \) that satisfies the conditions in Theorem 11.
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.

**Satisfiability of parametric formulae.** We consider the satisfiability of a set of parametric MITL formulae modified from [24, 13]. 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 [39, 14] for more details) with that in the continuous semantics (this article). For $k \geq 2$ and an interval $I$, let:

\[
F(k, I) = \bigwedge_{i=1}^{k} F_i^w p_i,
\]
\[
G(k, I) = \bigwedge_{i=1}^{k} G_i^w p_i,
\]
\[
U(k, I) = (\ldots (p_1 U_i^w p_2) U_i^w \ldots) U_i^w p_k,
\]
\[
R(k, I) = (\ldots (p_1 R_i^w p_2) R_i^w \ldots) R_i^w p_k,
\]

where $F_i^w$, $G_i^w$, 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. $F_{(5, \infty)} P \iff G_{(0, 5]} F P$). 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.

---

8 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.
Our approach w/o minimality

Table 2 Execution times for the satisfiability, validity and redundancy checks in [19].

<table>
<thead>
<tr>
<th>Formula</th>
<th>Our approach</th>
<th>Our approach w/o minimality</th>
<th>[19]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1 = F_{[0,30]} p_1 \land F_{[0,20]} p_1$</td>
<td>6.08s</td>
<td>7.50s</td>
<td>14s</td>
</tr>
<tr>
<td>$\phi_2 = F_{[0,30]} p_1 \Rightarrow G_{[0,20]} p_1$</td>
<td>3.08s</td>
<td>4.30s</td>
<td>7s</td>
</tr>
<tr>
<td>$\phi_3 = G_{[0,40]} p_1 \land G_{[0,40]} F_{[0,10]} p_1$</td>
<td>7.10s</td>
<td>38.15s</td>
<td>29s</td>
</tr>
<tr>
<td>$\phi_5 = F_{[0,40]} (p_1 \lor p_3) \land F_{[0,40]} F_{[0,10]} p_2 \land F_{[0,40]} G_{[0,30]} p_1$</td>
<td>12.04s</td>
<td>$&gt;1200s$</td>
<td>126s</td>
</tr>
</tbody>
</table>

Figure 4 The SA $A_{\text{lamp}}$. The transitions with solid tips reset clock $x$.

Validity and redundancy of specifications. We say that an MITL formula $\varphi$ is valid if $\neg\varphi$ is not satisfiable. If $\varphi$ is of the form $\bigwedge_{1 \leq i \leq k} \varphi_i$, we say that the conjunct $\varphi_i$ is redundant in $\varphi$ if the formula $\left(\bigwedge_{1 \leq j \leq k} \varphi_j \right) \Rightarrow \varphi_i$ is valid. In [19], 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 [19] in the table; since we do not impose a priori bounds on state changes (as opposed to [19]) and we use a much less powerful CPU, these numbers are not meant for direct comparison but rather for reference.

Model-checking a timed lamp. We consider a case study of a timed lamp from [9]. 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 [9], the system is given as an MITL formula (with past temporal operators) over atomic propositions $\{\ell, \text{on}, \text{off}\}$. While we can make use of projections to remove the past temporal operators [44, 28], it turned out that the resulting formula is too large. For this reason, we model the system directly as an $\text{SA} A_{\text{lamp}}$ (Figure 4). Then, via Proposition 10 and Theorem 11, we perform the same verification tasks as [9]: (1) checking the emptiness of $A_{\text{lamp}}$; (2) model-checking $A_{\text{lamp}}$ against $\varphi_1 = G_{[0,\infty]} \left( F_{[0,5]} (\neg \ell) \right)$, i.e. the lamp never stays lit for more than 5 time units; (3) model-checking $A_{\text{lamp}}$ against $\varphi_2 = F_{[0,\infty]} \left( G_{[0,5]} \ell \right) \Rightarrow F_{[0,\infty]} \left( \text{on} \land F_{[0,5]} \text{on} \right)$, 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 [9]. 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 [14]. 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 (via rewriting
Table 3 Execution times for the verification tasks in [9].

<table>
<thead>
<tr>
<th>Task</th>
<th>Our approach</th>
<th>Our approach w/o minimality</th>
<th>[9]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S(A_{lamp}) = \emptyset$</td>
<td>1.17s</td>
<td>–</td>
<td>4.24s</td>
</tr>
<tr>
<td>$S(A_{lamp} \times A_{\neg \varphi}) = \emptyset$</td>
<td>1.73s</td>
<td>1.77s</td>
<td>17.2s</td>
</tr>
<tr>
<td>$S(A_{lamp} \times A_{\neg \varphi'}) = \emptyset$</td>
<td>2.36s</td>
<td>13.18s</td>
<td>257.1s</td>
</tr>
</tbody>
</table>

or by components) and other temporal operators (such as those from ECL [28]). A possible future theoretical direction is to investigate whether the translation can be generalised (possibly with the techniques in [17] or [42]) to deal with signals that are not finitely-variable.

Acknowledgements. We thank the reviewers of this article for their helpful comments.

References


Timed-Automata-Based Verification of MITL over Signals


A Making signal automata bipartite

Proof of Proposition 2. For $A = (L, L_0, α, X, β, Δ, F)$, we define a corresponding $SA A_{bp} = (L_{bp}, L_{0, bp}, α_{bp}, X_{bp}, β_{bp}, Δ_{bp}, F_{bp})$ where

- $L_{bp} = \{ℓ*, ˆℓ, ℓo | ℓ ∈ L\}$;
- $L_{0, bp} = \{ℓ | ℓ ∈ L_0\}$;
- $α_{bp}(ℓ*) = α_{bp}( ˆℓ) = α_{bp}(ℓo) = α(ℓ)$ for every $ℓ ∈ L$;
- $X_{bp} = X ∪ \{y\}$ where $y$ is a fresh clock;
- $β_{bp}(ℓ*) = β_{bp}( ˆℓ) = β(ℓ) ∧ y = 0$, $β_{bp}(ℓo) = β(ℓ)$ for every $ℓ ∈ L$;
- $Δ_{bp} = \{(ℓ_1, λ, ℓ_2), ( ˆℓ_1, λ, ˆℓ_2), (ℓ_1, λ ∪ \{y\}, ℓ_2) | (ℓ_1, λ, ℓ_2) ∈ Δ\}$
  $∪ \{ℓ*, \{y\}, ˆℓ*, ℓo, ˆℓo | ℓ ∈ L\}$;
- $F_{bp} = \{ℓ*, ℓo | ℓ ∈ F\} | F ∈ F$.

Intuitively, we create three copies $ℓ*$, $ ˆℓ$, $ℓo$ of each location $ℓ$ of $A$ and use the clock $y$ to enforce the desired behaviour. In particular, the ‘dotted’ locations $ ˆℓ$ are used to deal with the situation where the ‘source’ interval is right-closed. One can verify that $A_{bp}$ is bipartite (let $L^s = \{ℓ*, ˆℓ | ℓ ∈ L\}$) and $S(A) = S(A_{bp})$. □
The component $C_{\chi}$ for $\chi = \psi_1 R \psi_2$ (Figure 5) is based on similar ideas as the component for $\psi_1 U \psi_2$. In this case, an obligation can be satisfied by either $\hat{\psi}_1 \land \hat{\psi}_2$ holding in a singular interval or $\hat{\psi}_1 \land * \psi_2$ holding in an open interval.

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