Mightyl: A compositional translation from mitl to timed automata

BRIHAYE, T, GEERAERTS, G, HO, Hsi-Ming and MONMEGE, B

Published version

BRIHAYE, T, GEERAERTS, G, HO, Hsi-Ming and MONMEGE, B (2017). Mightyl: A compositional translation from mitl to timed automata. Computer Aided Verification 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I, 10426, 421-440.

MightyL: A Compositional Translation from MITL to Timed Automata*

Thomas Brihaye[1], Gilles Geeraerts[2], Hsi-Ming Ho[1], 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. Metric Interval Temporal Logic (MITL) was first proposed in the early 1990s as a specification formalism for real-time systems. Apart from its appealing intuitive syntax, there are also theoretical evidences that make MITL a prime real-time counterpart of Linear Temporal Logic (LTL). Unfortunately, the tool support for MITL verification is still lacking to this day. In this paper, we propose a new construction from MITL to timed automata via very-weak one-clock alternating timed automata. Our construction subsumes the well-known construction from LTL to Büchi automata by Gastin and Oddoux and yet has the additional benefits of being compositional and integrating easily with existing tools. We implement the construction in our new tool MightyL and report on experiments using Uppaal and LTSmin as back-ends.

1 Introduction

The design of critical software that respect real-time specifications is a notoriously difficult problem. In this context, verification of programs against formal specifications is crucial, in order to handle the thin timing behaviours. In the untimed setting, a logic widely used both in academia and industry is *Linear Temporal Logic* (LTL) [32]. A crucial ingredient of its success is the possibility to translate LTL formulae into (Büchi) automata. In the real-time counterpart, *Metric Interval Temporal Logic* (MITL) [3] has been introduced twenty years ago where it was established that it can be translated into (Büchi) *timed automata* (TA). Beyond verification of real-time software, there are numerous interests in MITL from other domains, e.g. automated planning and scheduling [39], control engineering [18] and systems biology [6]. The translation from MITL to TAs is complicated and has led to some simplified constructions, e.g. [17, 28]. However, despite these efforts, the tool support for MITL is still lacking to this day. To the best of our knowledge, the only implementation of an automata-based construction is described in [10, 11], but is not publicly available. Since existing verification tools based on timed automata have been around for quite some time

* 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).

and have been successful (e.g. UPPAAL [27] first appeared in 1995), it would be preferable if such translation can be used with these tools.

In the present paper, we attempt to amend the situation by proposing a more practical construction from MITL to (Büchi) timed automata. Compared to [10, 11], our construction has the following advantages:

1. While we also use *one-clock alternating timed automata* (OCATA) [30] as an intermediate formalism, our construction exploits the 'very-weakness' of the structure of OCATAs obtained from MITL formulae to reduce state space. In particular, our construction subsumes LTL2BA [19] in the case of LTL.
2. The number of clocks in the resulting TA is reduced by a factor of up to two. This is achieved via a more fine-grained analysis of the possible clock values (see Section 5).
3. The construction is *compositional*: for each location of the OCATA $\mathcal{A}$ obtained from the input MITL formula, we construct a 'component' TA and establish a connection between the runs of $\mathcal{A}$ and the runs of the synchronous product of these components. Thanks to this connection, we can give the output TA in terms of components; this greatly simplifies the implementation, and speeds up its execution.
4. The construction is compatible with off-the-shelf model-checkers: our tool MIGHTYL generates output automata in the UPPAAL XML format which, besides UPPAAL [27] itself, can also be analysed by LTSMIN [24] with OPAAL front-end, TIAMO [9], ITS-TOOLS [35], DIVINE [5], etc.

Related work. There is already a number of MITL-to-TA constructions in the literature [3, 17, 28]. However, most of them interpret MITL over *signals* (i.e. the *continuous* semantics of MITL) and hence generate *signal automata*. This choice unfortunately hinders the possibility to leverage existing tools based on classical timed automata over *timed words* [2] and is probably one of the reasons why the aforementioned constructions have never been implemented.[4] We, following [4, 10, 11, 37] (among others), interpret MITL over timed words (i.e. the *pointwise* semantics of MITL). Note that there have been some implementations that deal with peculiar specification patterns over timed words (e.g. [1]). For MITL, apart from [10, 11] that we mentioned earlier, we are only aware of implementations for rather restricted cases, such as the safety fragment of $\mathsf{MITL}_{0,\infty}$ [12] or MITL over untimed words [39]. Our construction subsumes all of these approaches.

Using alternating automata as an intermediate formalism is a standard approach in LTL model-checking [36]. However, the translation from alternating automata to Büchi automata may incur an exponential blow-up if the output automaton is constructed explicitly [19]. For this reason, an *on-the-fly* approach is proposed in [21], but it requires a specialised model-checking algorithm. Alternatively, [8] gives a *symbolic* encoding of alternating automata which can be used directly with NUSMV [13], but minimality of transitions (which may

[4] Nonetheless, it has been argued that a continuous model of time is preferable from a theoretical point of view; see e.g. [22].

potentially improve the performance of verification algorithms, cf. [21]) is difficult to enforce in this setting (see also [14, 34]). Our construction combines the advantages of these approaches—it can be regarded as a symbolic encoding of OCATAs in TAs, enforcing some minimality criteria on transitions for efficiency (see Section 6)—and provides compatibility with existing tools that construct state spaces on-the-fly. By contrast, [17, 28], not based on OCATAs, also give the resulting automaton in terms of smaller component automata, but they have to use specialised product constructions to synchronise the components.

Apart from automata-theoretic approaches, [7] considers 'bounded model-checking' which encodes the satisfiability problem for MITL (in the continuous semantics) into an SMT problem (*Satisfiability Modulo Theories*) [15]. This approach is complete when very large bounds (numbers of regions of equivalent TA) are used, but such bounds are clearly impractical for current SMT solvers.

Outline. Section 2 starts with preliminary definitions of timed logics and (alternating) timed automata. Sections 3, 4 and 5 then give our new translation from formulae to generalised Büchi (timed) automata for LTL, $\mathsf{MITL}_{0,\infty}$ (a fragment of MITL where only intervals of the form $[0, a]$, $[0, a)$, $[a, +\infty)$, or $(a, +\infty)$ are allowed), and full MITL, respectively. We report on our OCaml implementation MightyL and some promising experiments on several benchmarks in Section 6.

2 Timed logics vs (alternating) timed automata

Timed languages. Let AP be a finite set of atomic propositions, and $\Sigma = 2^{\mathsf{AP}}$. A timed word over Σ is an infinite sequence $\rho = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots$ over $\Sigma \times \mathbb{R}^+$ with $(\tau_i)_{i \geq 1}$ a non-decreasing sequence of non-negative real numbers. We denote by $T\Sigma^\omega$ the set of timed words over Σ. A *timed language* is a subset of $T\Sigma^\omega$.

Timed logics. We consider the satisfiability and model-checking problem of Metric Interval Temporal Logic (MITL), an extension of Linear Temporal Logic (LTL) in which temporal operators can be labelled with *non-singular* timed intervals (or $[0, 0]$, which is the only singular interval we allow). Formally, MITL formulae over AP are generated by the grammar

$$\varphi := p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbf{X}_I \varphi \mid \varphi \, \mathbf{U}_I \, \varphi$$

where $p \in \mathsf{AP}$ and I is either a non-singular interval over $\mathbb{R}^+$ with endpoints in $\mathbb{N} \cup \{+\infty\}$ or $[0, 0]$. To simplify our explanations, we will only consider closed non-singular intervals in the sequel, i.e. intervals of the form $[a, b]$ or $[a, +\infty)$, with $0 \leq a < b < +\infty$. We let $|I|$ be the length of the interval I: $|[a, b]| = b - a$ for $0 \leq a < b < +\infty$ and $|[a, +\infty)| = +\infty$.

We consider the *pointwise semantics* and interpret MITL formulae over timed words. The semantics of a formula φ in MITL is defined inductively: given $\rho = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots \in T\Sigma^\omega$, and a position $i \geq 1$, we let

- $(\rho, i) \models p$ if $p \in \sigma_i$;
- $(\rho, i) \models \varphi_1 \wedge \varphi_2$ if $(\rho, i) \models \varphi_1$ and $(\rho, i) \models \varphi_2$;

– $(\rho, i) \models \neg\varphi$ if $(\rho, i) \not\models \varphi$;
– $(\rho, i) \models \mathbf{X}_I \varphi$ if $(\rho, i+1) \models \varphi$ and $\tau_{i+1} - \tau_i \in I$;
– $(\rho, i) \models \varphi_1 \, \mathbf{U}_I \, \varphi_2$ if there exists $j \geq i$, $(\rho, j) \models \varphi_2$, $\tau_j - \tau_i \in I$, and, for all $i \leq k < j$, $(\rho, k) \models \varphi_1$.

We derive other Boolean operators with the following macros: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\top \equiv p \vee \neg p$, $\bot \equiv \neg\top$, and $\varphi_1 \Rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$. We also define other temporal operators as usual: the 'eventually' operator $\mathbf{F}_I \varphi \equiv \top \, \mathbf{U}_I \, \varphi$, the 'globally' operator $\mathbf{G}_I \varphi \equiv \neg\mathbf{F}_I \neg\varphi$, the 'release' operator $\varphi_1 \, \mathbf{R}_I \, \varphi_2 \equiv \neg((\neg\varphi_1) \, \mathbf{U}_I \, (\neg\varphi_2))$, and the 'dual-next' operator $\overline{\mathbf{X}}_I \varphi \equiv \neg\mathbf{X}_I \neg\varphi$ (contrary to LTL, it is not true that $\neg\mathbf{X}_I \varphi \equiv \mathbf{X}_I \neg\varphi$). With the release and dual-next operators, we can transform every formula φ into *negative normal form*, i.e. formulae using only predicates of AP, their negations, and the operators $\vee$, $\wedge$, $\mathbf{U}_I$, $\mathbf{R}_I$, $\mathbf{X}_I$, and $\overline{\mathbf{X}}_I$. To help the understanding, let us detail the semantics of $\varphi_1 \, \mathbf{R}_I \, \varphi_2$:

– $(\rho, i) \models \varphi_1 \, \mathbf{R}_I \, \varphi_2$ if for all $j \geq i$ such that $\tau_j - \tau_i \in I$, either $(\rho, j) \models \varphi_2$, or there exists $i \leq k < j$ such that $(\rho, k) \models \varphi_1$.

We say that ρ satisfies the formula φ, written $\rho \models \varphi$ if $(\rho, 1) \models \varphi$, and we denote by $[\![\varphi]\!]$ the set of all timed words satisfying φ. When writing formulae, we omit the trivial interval $[0, +\infty)$. LTL is the fragment of MITL where all operators are labelled by $[0, \infty)$; and $\mathsf{MITL}_{0,\infty}$ is the fragment where, in all intervals, either the left endpoint is 0 or the right endpoint is $+\infty$.

Timed automata. Let X be a finite set of real valued variables, called clocks. The set $\mathcal{G}(X)$ of *clock constraints* g over X is defined by $g := \top \mid g \wedge g \mid x \bowtie c$, where $\bowtie \in \{\leq, <, \geq, >\}$, $x \in X$ and $c \in \mathbb{N}$. A *valuation* over X is a mapping $v \colon X \to \mathbb{R}^+$. We denote by $\mathbf{0}$ the valuation that maps every clock to 0, and we write the valuation simply as a value in $\mathbb{R}^+$ when X is a singleton. The satisfaction of a constraint g by a valuation v is defined in the usual way and noted $v \models g$, and we denote by $[\![g]\!]$ the set of valuations v satisfying g. For $t \in \mathbb{R}^+$, we let $v + t$ be the valuation defined by $(v + t)(x) = v(x) + t$ for all $x \in X$. For $R \subseteq X$, we let $v[R \leftarrow 0]$ be the valuation defined by $(v[R \leftarrow 0])(x) = 0$ if $x \in R$, and $(v[R \leftarrow 0])(x) = v(x)$ otherwise.

We introduce the notion of *generalised Büchi timed automaton* (GBTA) as an extension of classical timed automata [2] with a generalised acceptance condition (used by [20] in the untimed setting). A GBTA is a tuple $\mathcal{A} = (L, \Sigma, \ell_0, \Delta, \mathcal{F})$ where L is a finite set of locations, Σ is a finite alphabet, $\ell_0 \in L$ is the initial location, $\Delta \subseteq L \times \Sigma \times \mathcal{G}(X) \times 2^X \times L$ is the transition relation, and $\mathcal{F} = \{F_1, \ldots, F_n\}$, with $F_i \subseteq L$ for all $1 \leq i \leq n$, is the set of sets of final locations. A timed automaton (TA), as described in [2], is a special case of GBTA where $\mathcal{F} = \{F\}$ is a singleton (F contains the accepting locations of the TA). A *state* of $\mathcal{A}$ is a pair (ℓ, v) of a location $\ell \in L$ and a valuation v of the clocks in X. A *run* of $\mathcal{A}$ over the timed word $(\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots \in T\Sigma^\omega$ is a sequence of states $C_0, C_1, \ldots$ where (i) $C_0 = (\ell_0, \mathbf{0})$ and (ii) for each $i \geq 0$ such that $C_i = (\ell, v)$, there is a transition $(\ell, \sigma_{i+1}, g, R, \ell')$ such that $C_{i+1} = (\ell', v')$, $v + (\tau_{i+1} - \tau_i) \models g$ (assuming $\tau_0 = 0$) and $v' = (v + (\tau_{i+1} - \tau_i))[R \leftarrow 0]$. By the generalised Büchi

acceptance condition, a run is *accepting* if and only if the set of locations that it visits infinitely often contains at least one location from each set F_i, for all $1 \leq i \leq n$. We let $[\![\mathcal{A}]\!]$ be the set of timed words on which there exist accepting runs of $\mathcal{A}$.

Synchronisation of timed automata. In the following, we will consider GBTAs described by synchronous products of several components. More precisely, given two GBTAs $\mathcal{A}^1 = (L^1, \Sigma, \ell_0^1, \Delta^1, \mathcal{F}^1)$ and $\mathcal{A}^2 = (L^2, \Sigma, \ell_0^2, \Delta^2, \mathcal{F}^2)$ over disjoint sets of clocks, we define the GBTA $\mathcal{A}^1 \times \mathcal{A}^2 = (L, \Sigma, \ell_0, \Delta, \mathcal{F})$ obtained by synchronising $\mathcal{A}^1$ and $\mathcal{A}^2$. Its set of locations is $L = L^1 \times L^2$, with $\ell_0 = (\ell_0^1, \ell_0^2)$. The acceptance condition is obtained by mimicking a disjoint union of the generalised Büchi conditions: assuming $\mathcal{F}^1 = \{F_1, \ldots, F_n\}$ and $\mathcal{F}^2 = \{G_1, \ldots, G_m\}$, we let $\mathcal{F} = \{F_1 \times L^2, \ldots, F_n \times L^2, L^1 \times G_1, \ldots, L^1 \times G_m\}$. Finally, $((\ell_1^1, \ell_1^2), \sigma, g, R, (\ell_2^1, \ell_2^2)) \in \Delta$ if there exists $(\ell_1^1, \sigma, g^1, R^1, \ell_2^1) \in \Delta^1$ and $(\ell_1^2, \sigma, g^2, R^2, \ell_2^2) \in \Delta^2$ such that $g = g^1 \wedge g^2$ and $R = R^1 \cup R^2$. This definition can be extended for the synchronisation of a set of GBTAs $\{\mathcal{A}^i \mid i \in I\}$: the product is then written as $\prod_{i \in I} \mathcal{A}^i$.

One-clock alternating timed automata. One-clock alternating timed automata (OCATA) [30] extend (non-deterministic) one-clock timed automata by adding *conjunctive transitions*. Intuitively, a conjunctive transition spawns several copies of the automaton that run in parallel from the targets of the transition. A word is accepted if and only if *all* copies accept it. An example is shown in Fig. 1, where the conjunctive transition is the hyperedge starting from ℓ_0.

Formally, we consider a single clock x and, for a set L of locations, let $\Gamma(L)$ be the set of formulae defined by

$$\gamma := \top \mid \bot \mid \gamma \vee \gamma \mid \gamma \wedge \gamma \mid \ell \mid x \bowtie c \mid x.\gamma$$

where $c \in \mathbb{N}$, $\bowtie \in \{\leq, <, \geq, >\}$, and $\ell \in L$. Compared to the clock constraints defined above for TAs, $\Gamma(L)$ allows non-determinism ($\vee$ operator), locations as atoms, and expressions of the form $x.\gamma$ (meaning that x is reset in γ). An OCATA is a tuple $\mathcal{A} = (L, \Sigma, \ell_0, \delta, F)$ where L is a finite set of locations, Σ is a finite alphabet, $\ell_0 \in L$ is the initial location, $\delta \colon L \times \Sigma \to \Gamma(L)$ is the transition function, and $F \subseteq L$ is the set of final locations. A *state* of $\mathcal{A}$ is a pair (ℓ, v) of a location in L and a valuation of the single clock x. Models of the formulae in $\Gamma(L)$, with respect to a clock valuation $v \in \mathbb{R}^+$, are sets of states M:

- $M \models_v \top$; $M \models_v \ell$ if $(\ell, v) \in M$; $M \models_v x \bowtie c$ if $v \bowtie c$; $M \models_v x.\gamma$ if $M \models_0 \gamma$;
- $M \models_v \gamma_1 \wedge \gamma_2$ if $M \models_v \gamma_1$ and $M \models_v \gamma_2$;
- $M \models_v \gamma_1 \vee \gamma_2$ if $M \models_v \gamma_1$ or $M \models_v \gamma_2$.

A set M of states is said to be a *minimal model* of the formula $\gamma \in \Gamma(S)$ with respect to a clock valuation $v \in \mathbb{R}^+$ if and only if $M \models_v \gamma$ and there is no proper subset $M' \subset M$ with $M' \models_v \gamma$. A run of $\mathcal{A}$ over a timed word $\rho = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \cdots \in T\Sigma^\omega$ is a rooted directed acyclic graph (DAG) $G = (V, \to)$ with vertices of the form $(\ell, v, i) \in L \times \mathbb{R}^+ \times \mathbb{N}$, $(\ell_0, 0, 0)$ as root, and edges as follows: for every vertex (ℓ, v, i), we choose a minimal model M of the formula

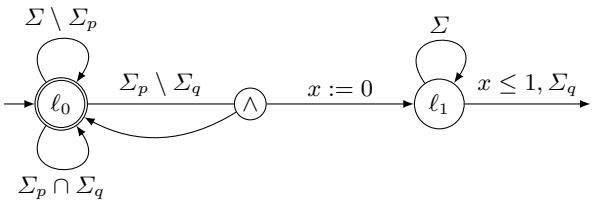

Fig. 1. An OCATA accepting the language of $\mathbf{G}(p \Rightarrow \mathbf{F}_{[0,1]}q)$.

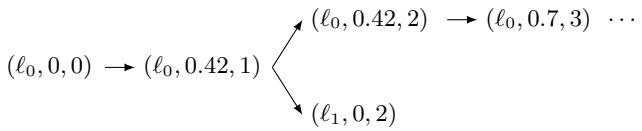

Fig. 2. A run of the OCATA of Fig. 1 over $(\emptyset, 0.42)(\{p\}, 0.42)(\{q\}, 0.7)\cdots$.

$\delta(\ell, \sigma_{i+1})$ with respect to $v + (\tau_{i+1} - \tau_i)$ (again, $\tau_0 = 0$), and we have an edge $(\ell, v, i) \to (\ell', v', i+1)$ in G for every state (ℓ', v') appearing in model M. Such a run is *accepting* if and only if there is no infinite path in G that visit final locations only finitely often. We let $[\![\mathcal{A}]\!]$ be the set of timed words on which there exist accepting runs of $\mathcal{A}$.

It is also useful to see a run as a linear sequence of *configurations* (i.e. finite sets of states) which gather all states at a given DAG level. Formally, from a DAG $G = (V, \to)$ we extract the sequence of configurations $K_0, K_1, \ldots$ where $K_i = \{(\ell, v) \mid (\ell, v, i) \in V\}$ for all $i \geq 0$.[5]

Example 1. Consider the OCATA of Fig. 1 on the alphabet $\Sigma = 2^{\{p,q\}}$. For each proposition $\pi \in \{p, q\}$, we write $\Sigma_\pi = \{\sigma \in \Sigma \mid \pi \in \sigma\}$. A run over the timed word $(\emptyset, 0.42)(\{p\}, 0.42)(\{q\}, 0.7)\cdots$ is depicted in Fig. 2. It starts with the DAG rooted in $(\ell_0, 0, 0)$ (initially, there is only one copy in ℓ_0 with the clock equal to 0). This root has a single successor $(\ell_0, 0.42, 1)$, which has two successors $(\ell_0, 0.42, 2)$ and $(\ell_1, 0, 2)$ (after firing the conjunctive transition from ℓ_0). Then, $(\ell_1, 0, 2)$ has no successor since the empty model is a minimal model of the next transition (the transition from ℓ_1 points to no location). The associated sequence of configurations starts by: $\{(\ell_0, 0)\}, \{(\ell_0, 0.42)\}, \{(\ell_0, 0.42), (\ell_1, 0)\}\cdots$

Each formula φ of MITL can be translated into an OCATA $\mathcal{A}_\varphi$ that accepts the same language [11,30], and with a number of locations linear in the number of subformulae of φ. We recall the definition of $\mathcal{A}_\varphi$ for the sake of completeness. The set of locations of $\mathcal{A}_\varphi$ contains: (i) φ_{init}; (ii) all the subformulae of φ (that we suppose to be in negative normal form) whose outermost operator is $\mathbf{U}_I$ or $\mathbf{R}_I$; and (iii) ψ^r for each subformulae ψ of φ whose outermost operator is $\mathbf{X}_I$

[5] In the current (infinite-word) setting, we cannot define acceptance conditions in terms of configurations as in [30].

or $\overline{\mathbf{X}}_I$. Its initial location is φ_{init}, and the accepting locations of F are all the subformulae of the form $\varphi_1 \mathbf{R}_I \varphi_2$. Finally, δ is defined inductively:

- $\delta(\varphi_{init}, \sigma) = x.\delta(\varphi, \sigma)$, $\delta(\top, \sigma) = \top$, and $\delta(\bot, \sigma) = \bot$;
- $\delta(p, \sigma) = \top$ if $p \in \sigma$, $\delta(p, \sigma) = \bot$ otherwise;
- $\delta(\neg p, \sigma) = \top$ if $p \notin \sigma$, $\delta(\neg p, \sigma) = \bot$ otherwise;
- $\delta(\varphi_1 \vee \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \vee \delta(\varphi_2, \sigma)$, and $\delta(\varphi_1 \wedge \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \wedge \delta(\varphi_2, \sigma)$;
- $\delta(\varphi_1 \mathbf{U}_I \varphi_2, \sigma) = (x.\delta(\varphi_2, \sigma) \wedge x \in I) \vee (x.\delta(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U}_I \varphi_2 \wedge x \leq \sup I)$;
- $\delta(\varphi_1 \mathbf{R}_I \varphi_2, \sigma) = (x.\delta(\varphi_2, \sigma) \vee x \notin I) \wedge (x.\delta(\varphi_1, \sigma) \vee \varphi_1 \mathbf{R}_I \varphi_2 \vee x > \sup I)$;
- $\delta(\mathbf{X}_I \varphi, \sigma) = x.(\mathbf{X}_I \varphi)^r$, and $\delta((\mathbf{X}_I \varphi)^r, \sigma) = x \in I \wedge x.\delta(\varphi, \sigma)$;
- $\delta(\overline{\mathbf{X}}_I \varphi, \sigma) = x.(\overline{\mathbf{X}}_I \varphi)^r$, and $\delta((\overline{\mathbf{X}}_I \varphi)^r, \sigma) = x \notin I \vee x.\delta(\varphi, \sigma)$.

As already noticed in [11], the OCATA $\mathcal{A}_\varphi$ produced from an MITL formula φ is *very-weak* [19, 26, 29], i.e. it comes with a partial order on its locations such that all locations appearing in $\delta(\ell, \sigma)$ are bounded above by ℓ in this order. For an OCATA $\mathcal{A}_\varphi$ obtained from an MITL formula φ, the order is given by the subformula order: φ_{init} is the greatest element in the order, and a location ψ is less than χ if ψ is a subformula of χ. We will also make use of the following properties of δ: (i) if ℓ' appears in $\delta(\ell, \sigma)$ then it is preceded by a clock reset if and only if $\ell' \neq \ell$; and (ii) each ℓ' either has no parent or has a unique parent, i.e. there is a unique $\ell \neq \ell'$ such that ℓ' appears in $\delta(\ell, \sigma)$ for some σ.

Theorem 2 ([11]). *For all formulae φ of* MITL, $[\![\mathcal{A}_\varphi]\!] = [\![\varphi]\!]$.

Remark 3. To ease the presentation, we use Boolean formulae over atomic propositions as transition labels. For instance, $\Sigma \setminus \Sigma_p$ will be written as $\neg p$.

3 Compositional removal of alternation

The current and next two sections are devoted to explaining the core idea of our construction: simulate the OCATA $\mathcal{A}_\varphi$ obtained from an MITL formula φ by the synchronous product of component Büchi timed automata, one for each temporal subformula (i.e. a subformula whose outermost operator is temporal). The very-weakness of $\mathcal{A}_\varphi$ is crucial for our construction to work: a run of $\mathcal{A}_\varphi$ is accepting if and only if $\mathcal{A}_\varphi$ does not get stuck at a non-accepting location in any branch. Therefore, we can keep track of each location with a separate component and simply define a suitable Büchi acceptance condition on each such component.[6] Our compositional construction preserves the structure of the formula, and thus we can hope that the model-checking tool (which is responsible for the composition) takes this into account.[7] At the very least, the model-checking tool can use an on-the-fly approach in composition (as is indeed the

[6] This is not possible for general (not very-weak) OCATAs since it might be the case that a branch alternates between several non-accepting location without ever hitting an accepting location.

[7] The same idea underlies the antichain-based algorithms for LTL model-checking [38], where the structure can be exploited to define a pre-order on the state space of the resulting automaton.

case for UPPAAL and LTSMIN), which is often faster in practice: the explicit construction of the whole product can be avoided when there is an accepting run.

In what follows, let φ be an MITL formula over AP in negative normal form and $\mathcal{A}_\varphi$ be the OCATA obtained from φ with the translation described earlier. For the sake of simplicity, we make the following assumptions:

- $\mathbf{X}_I$ and $\overline{\mathbf{X}}_I$ do not appear in φ;
- each temporal subformula ψ of φ appears only once in φ.

Let Φ be the set of temporal subformulae of φ. We introduce a new atomic proposition p_ψ for each subformula $\psi \in \Phi$ (i.e. for each non-initial location of the OCATA $\mathcal{A}_\varphi$) and let AP_φ be the set of these new atomic propositions. For each (not necessarily temporal) subformula ψ of φ, we denote by $\mathcal{P}_\psi$ the set of atomic propositions $p_\xi \in \mathsf{AP}_\varphi$ such that ξ is a top-level temporal subformula of ψ, i.e. the outermost operator of ξ is $\mathbf{U}_I$ or $\mathbf{R}_I$, yet ξ does not occur under the scope of another $\mathbf{U}_I$ or $\mathbf{R}_I$. For instance, $\mathcal{P}_{p\mathbf{U}_I q \vee r\mathbf{U}_I(s\mathbf{R}t)} = \{p_{p\mathbf{U}_I q}, p_{r\mathbf{U}_I(s\mathbf{R}t)}\}$.

Hintikka sequences and triggers. A *Hintikka sequence* of φ is a timed word ρ' over $2^{\mathsf{AP}\cup\mathsf{AP}_\varphi}$. Intuitively, Hintikka sequences can be regarded as an instrumented version of timed words, where the extra atomic propositions from AP_φ are *triggers* that connect timed words to their runs in the OCATA $\mathcal{A}_\varphi$; this is the central notion of our construction which, as we will prove, indeed simulates the runs of $\mathcal{A}_\varphi$. Pulling the trigger p_ψ (i.e. setting p_ψ to true) at some point means that ψ is required to hold at this point. However, the absence of a trigger p_ξ does not mean that subformula ξ must not be satisfied—its satisfaction is simply not required at this point. We denote by $\mathsf{proj}_{\mathsf{AP}}(\rho')$ the timed word obtained by hiding all the atomic propositions in AP_φ from ρ'. We also let $\mathsf{proj}_{\mathsf{AP}}(\mathcal{L}) = \{\mathsf{proj}_{\mathsf{AP}}(\rho') \mid \rho' \in \mathcal{L}\}$ for a timed language $\mathcal{L}$ over $2^{\mathsf{AP}\cup\mathsf{AP}_\varphi}$.

Formulae over $\mathsf{AP} \cup \mathsf{AP}_\varphi$. We now introduce some syntactic operations on Boolean combinations of atomic propositions in $\mathsf{AP} \cup \mathsf{AP}_\varphi$, that will be used to construct the component Büchi automata later. Specifically, for a subformula ψ of φ, we define formulae $\overline{\psi}$, $*\psi$, $\sim\psi$, and $\widehat{\psi}$.

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

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

The formula $*\psi$, read as "do not pull the triggers of ψ", will be used to ensure that our component automata only follow the minimal models of the transition function of $\mathcal{A}_\varphi$ (we will see in Section 6 how crucial it is, for performance, to generate only minimal models). It is the conjunction of negations of all the atomic propositions in $\mathcal{P}_\psi$. As a concrete example,

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

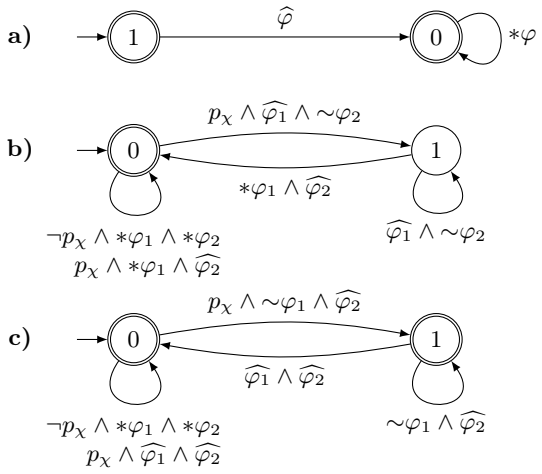

Fig. 3. The automata a) $\mathcal{C}_{init}$ and $\mathcal{C}_\chi$ for b) $\chi = \varphi_1 \,\mathbf{U}\, \varphi_2$, and c) $\chi = \varphi_1 \,\mathbf{R}\, \varphi_2$.

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

$$\mathsf{mm}(p) = p \qquad \mathsf{mm}(\neg p) = \neg p \qquad \mathsf{mm}(\top) = \top \qquad \mathsf{mm}(\bot) = \bot$$
$$\mathsf{mm}(\alpha_1 \vee \alpha_2) = \big(\mathsf{mm}(\alpha_1) \wedge \sim\alpha_2\big) \vee \big(\mathsf{mm}(\alpha_2) \wedge \sim\alpha_1\big) \vee \big((\alpha_1 \vee \alpha_2) \wedge *\alpha_1 \wedge *\alpha_2\big)$$
$$\mathsf{mm}(\alpha_1 \wedge \alpha_2) = \mathsf{mm}(\alpha_1) \wedge \mathsf{mm}(\alpha_2)\,.$$

Intuitively, $\mathsf{mm}(\alpha)$ is satisfiable if and only if α is satisfiable, but $\mathsf{mm}(\alpha)$ only permits models of α that are minimal with respect to the triggers it contains: for $\mathsf{mm}(\alpha_1 \vee \alpha_2)$ to be true, either $\mathsf{mm}(\alpha_1)$ is true and α_2 does not hold, or vice versa, or $\alpha_1 \vee \alpha_2$ is indeed true, but not because of any of the triggers it contains.

Component Büchi automata for LTL. We are now ready to present the construction for the case that φ is an LTL formula. Instead of building a monolithic Büchi automaton $\mathcal{B}_\varphi$ directly from the alternating automaton, as in [19], we build small component Büchi automata that are language-equivalent to the automaton $\mathcal{B}_\varphi$, once synchronised. There is an initial component $\mathcal{C}_{init}$, and a component Büchi automaton $\mathcal{C}_\chi$, for each $\chi \in \Phi$ (see Fig. 3). Consider, for instance, the case $\chi = \varphi_1 \,\mathbf{U}\, \varphi_2$. Component $\mathcal{C}_\chi$ has two locations 0 and 1 with the following intended meaning: $\mathcal{C}_\chi$ is in location 1 if and only if the trigger p_χ has been pulled in the past by $\mathcal{C}_{init}$, in which case $p_\chi \in \mathcal{P}_\varphi$, or by a unique component $\mathcal{C}_{\psi_1 \mathbf{U}_I \psi_2}$ (or $\mathcal{C}_{\psi_1 \mathbf{R}_I \psi_2}$) such that $p_\chi \in \mathcal{P}_{\psi_1}$ or $p_\chi \in \mathcal{P}_{\psi_2}$, and χ has not been satisfied yet. When component $\mathcal{C}_\chi$ is in location 1, we say that we have an *obligation* for χ. To satisfy this obligation, we must see a letter in the future where φ_2 holds. Thus, there is a self-loop on location 1 whose label ensures that φ_2 does not hold (because of $\sim\varphi_2$), while φ_1 still holds (this is ensured by $\widehat{\varphi_1}$,

which also pulls a minimal set of triggers for φ_1 to be satisfied). $\mathcal{C}_\chi$ moves back from 1 to 0 when φ_2 holds, while no trigger of φ_1 should be pulled at this instant (which is translated by $*\varphi_1$). From location 0, if we do not read trigger p_χ, nothing has to be checked and we do not pull any trigger. However, if p_χ is pulled, then, either φ_2 holds right away and the obligation is fulfilled immediately, or we jump to location 1. The component $\mathcal{C}_\chi$ for the case $\chi = \varphi_1 \,\mathbf{R}\, \varphi_2$ is based on a similar reasoning. We state the following proposition without proof as it will be superseded by a stronger proposition in the next section.

Proposition 4. *For all* LTL *formulae* φ, $\mathsf{proj}_{\mathsf{AP}}(\llbracket \mathcal{C}_{init} \times \prod_{\chi \in \Phi} \mathcal{C}_\chi \rrbracket) = \llbracket \varphi \rrbracket$.

Example 5. Consider the LTL formula $\mathbf{G}(p \Rightarrow \mathbf{F}q)$ that can be rewritten into negative normal form as $\varphi = \bot \,\mathbf{R}\, (\neg p \vee \top \,\mathbf{U}\, q)$. Then, the three component Büchi automata $\mathcal{C}_{init}$, $\mathcal{C}_\varphi$ and $\mathcal{C}_{\mathbf{F}q}$, after the constraints on the transitions are simplified, are depicted on the top of Fig. 4, The automaton $\mathcal{C} = \mathcal{C}_{init} \times \mathcal{C}_\varphi \times \mathcal{C}_{\mathbf{F}q}$ is depicted in the middle of the figure. Once atomic propositions in AP_φ are projected away, one obtains an automaton isomorphic to the one at the bottom of the figure that accepts $\llbracket \varphi \rrbracket$.

4 The case of $\mathsf{MITL}_{0,\infty}$

We now describe how to lift the translation we described earlier to the timed operators of $\mathsf{MITL}_{0,\infty}$. The new components for $\mathbf{U}_{[0,a]}$, $\mathbf{R}_{[0,a]}$, and $\mathbf{R}_{[a,\infty)}$ are depicted in Fig. 5. They have the same shape as the components for untimed $\mathbf{U}$ and $\mathbf{R}$ (see Fig. 3); only the guards are changed to reflect the more involved semantics of the timed operators. Observe that these automata have only one clock. To understand why this is sufficient, consider the formula $\mathbf{G}(p \Rightarrow \chi)$ with $\chi = p\mathbf{U}_{[0,2]}\, q$. After reading $(\{p\}, 0)(\{p\}, 0.4)(\{p\}, 1)$, the OCATA $\mathcal{A}_\varphi$ reaches the configuration $\{(\varphi, 0), (\chi, 0), (\chi, 0.6), (\chi, 1)\}$, meaning intuitively that, to satisfy the formula, one must fulfil three obligations related to χ: to see q's within 2, 1.4, and 1 time units, respectively. Hence, we can store the earliest obligation, corresponding to $(\chi, 1)$, only (as already observed in [11]). Indeed, if the corresponding instance of χ is satisfied, it means that there will be a q occurring within less that 1 time unit, which will also satisfy all the other obligations. More generally, for operators $\mathbf{U}_{[0,a]}$ and $\mathbf{R}_{[a,\infty)}$, it is always the case that only the oldest obligation has to be stored, while for operators $\mathbf{R}_{[0,a]}$ and $\mathbf{U}_{[a,\infty)}$, only the earliest obligation has to be stored. This is translated in the components by the absence/presence of resets on transitions that leave state 1 (which is reached when an obligation is currently active) and read p_χ.

For $\chi = \varphi_1 \,\mathbf{U}_{[a,\infty)}\, \varphi_2$, the situation is slightly more complicated, although one clock is again sufficient. The corresponding component is in Fig. 6 and has four locations. To understand why, consider the case when there is an obligation for χ associated with the current valuation $v \geq a$ of clock x ($\mathcal{C}_\chi$ is in location 1), the current letter contains p_χ and satisfies both $\widehat{\varphi_1}$ and $\widehat{\varphi_2}$. Since the trigger has been pulled, $\mathcal{C}_\chi$ should stay in the non-accepting location 1. On the other

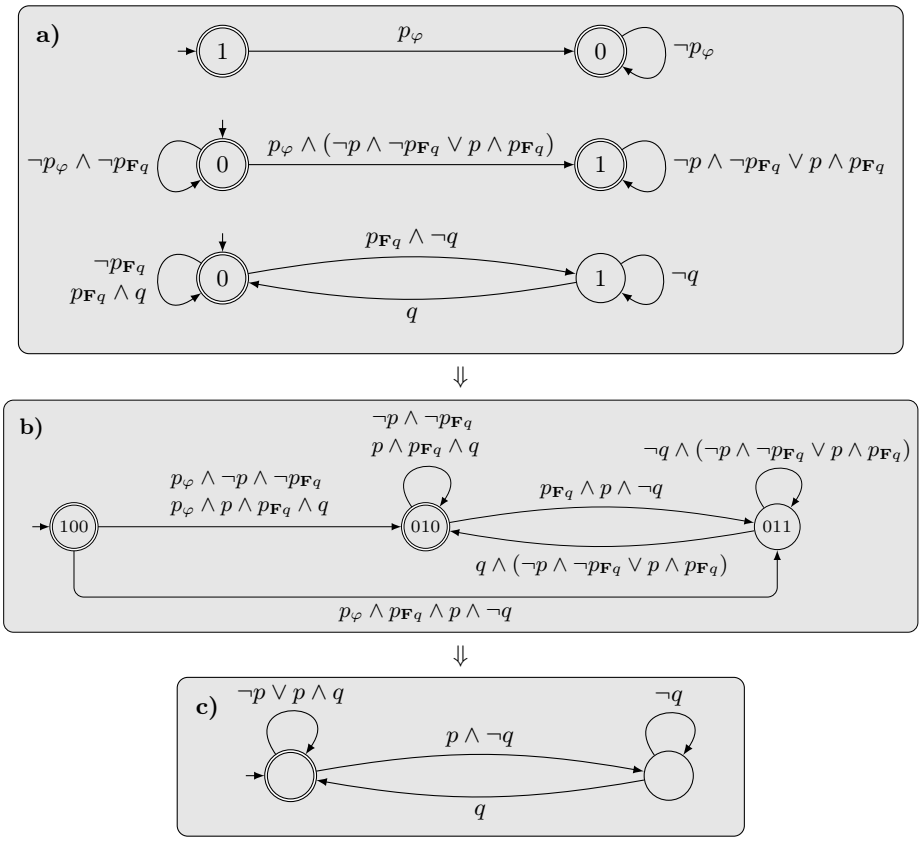

Fig. 4. a) Component Büchi automata for the formula $\varphi = \bot \mathbf{R} (\neg p \vee \top \mathbf{U}\, q)$; b) Büchi automaton obtained by the product of the components; c) Büchi automaton obtained by projecting away AP_φ (and merging two identical locations).

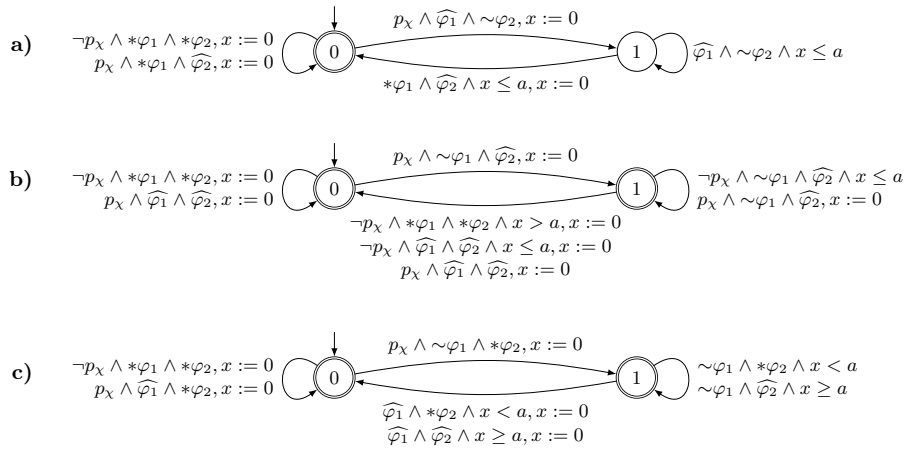

Fig. 5. One-clock TA for the subformulae: a) $\chi = \varphi_1 \,\mathbf{U}_{[0,a]}\, \varphi_2$, b) $\chi = \varphi_1 \,\mathbf{R}_{[0,a]}\, \varphi_2$, and c) $\chi = \varphi_1 \,\mathbf{R}_{[a,\infty)}\, \varphi_2$.

hand, the pending obligation has also been fulfilled, and an accepting location should be visited. So, instead of staying in 1, $\mathcal{C}_\chi$ moves to $1'$ in this case: $1'$ is a copy of 1 as far as transitions are concerned, but it is accepting. The location $1''$ is used to deal with the situation where p_χ is launched infinitely often but no two occurrences of p_χ are separated by more than a time units; in this case, we non-deterministically move to $1''$ and add a new obligation (by resetting x) after the current obligation has been verified. Notice that this problem cannot occur for $\varphi_1 \,\mathbf{U}\, \varphi_2$, or $\varphi_1 \,\mathbf{U}_{[0,a]}\, \varphi_2$: in these cases, the new obligation is immediately fulfilled, and the automaton moves to the initial, accepting, location.

We now present the extension of Proposition 4 to the case of $\mathsf{MITL}_{0,\infty}$. The proof relies on a function that, given a formula $\gamma = \delta(\ell, \sigma)$ (where δ is the transition function of $\mathcal{A}_\varphi$, ℓ is a location of $\mathcal{A}_\varphi$, and $\sigma \in \Sigma = 2^{\mathsf{AP}}$) and a minimal model M of γ with respect to a clock valuation $v \in \mathbb{R}^+$, recovers the set of triggers activated. Formally, we write $\mathsf{trig}_\varphi(M, \gamma, v)$ for the subset of AP_φ inductively defined by (the rule for $x.\gamma$ where $\gamma = \delta(\ell, \sigma)$ for some $\ell \in \Phi$ has precedence over the rule for $x.(\gamma_1 \wedge \gamma_2)$ and $x.(\gamma_1 \vee \gamma_2)$):

- $\mathsf{trig}_\varphi(M, \gamma_1 \wedge \gamma_2, v) = \mathsf{trig}_\varphi(M, \gamma_1, v) \cup \mathsf{trig}_\varphi(M, \gamma_2, v)$;
- $\mathsf{trig}_\varphi(M, \gamma_1 \vee \gamma_2, v) = \begin{cases} \mathsf{trig}_\varphi(M, \gamma_1, v) & \text{if } M \models_v \gamma_1 \\ \mathsf{trig}_\varphi(M, \gamma_2, v) & \text{otherwise;} \end{cases}$
- $\mathsf{trig}_\varphi(M, x.\gamma, v) = \{p_\ell\} \cup \mathsf{trig}_\varphi(M, \gamma, 0)$ if $\gamma = \delta(\ell, \sigma)$ for some $\ell \in \Phi$;
- $\mathsf{trig}_\varphi(M, x.(\gamma_1 \wedge \gamma_2), v) = \mathsf{trig}_\varphi(M, x.\gamma_1, v) \cup \mathsf{trig}_\varphi(M, x.\gamma_2, v)$;
- $\mathsf{trig}_\varphi(M, x.(\gamma_1 \vee \gamma_2), v) = \begin{cases} \mathsf{trig}_\varphi(M, x.\gamma_1, v) & \text{if } M \models_v x.\gamma_1 \\ \mathsf{trig}_\varphi(M, x.\gamma_2, v) & \text{otherwise;} \end{cases}$
- $\mathsf{trig}_\varphi(M, \gamma, v) = \emptyset$ otherwise.

Proposition 6. *For all* $\mathsf{MITL}_{0,\infty}$ *formulae* φ, $\mathsf{proj}_{\mathsf{AP}}([\![\mathcal{C}_{init} \times \prod_{\chi \in \Phi} \mathcal{C}_\chi]\!]) = [\![\varphi]\!]$.

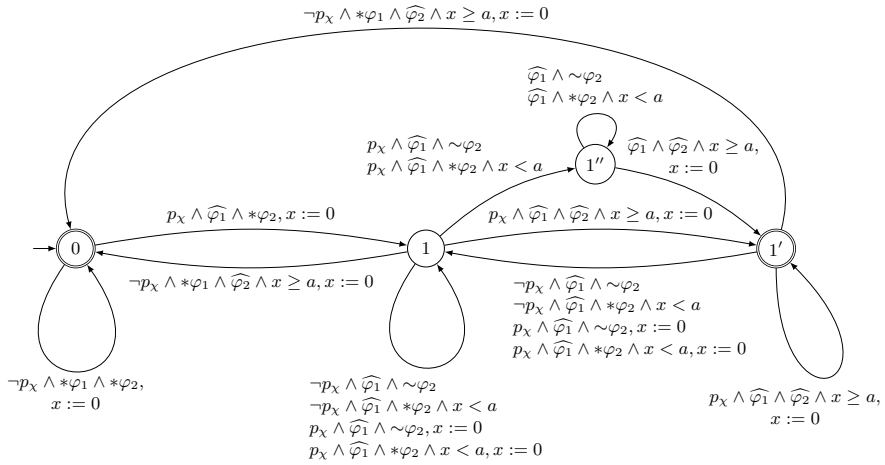

Fig. 6. One-clock TA for the subformula $\chi = \varphi_1 \, \mathbf{U}_{[a,\infty)} \, \varphi_2$.

Proof (Sketch). Recall that Theorem 2 states that $[\![\varphi]\!] = [\![\mathcal{A}_\varphi]\!]$. Therefore, it suffices to relate the accepting runs of the synchronous product of all component Büchi timed automata $\mathcal{C} = \mathcal{C}_{init} \times \prod_{\chi \in \Phi} \mathcal{C}_\chi$ with the accepting runs of $\mathcal{A}_\varphi$. Let us consider a timed word $\rho \in [\![\mathcal{A}_\varphi]\!]$ and an accepting run $G = (V, \to)$ of $\mathcal{A}_\varphi$ over ρ. Let $K_0, K_1, \ldots$ be the sequence of configurations associated with G.

We first construct the instrumented timed word ρ' over $2^{\mathsf{AP} \cup \mathsf{AP}_\varphi}$ from ρ and G by adding the triggers in AP_φ according to the minimal models selected in G. More precisely, for all $i \geq 0$, we associate with every state (ℓ, v) of K_i the pair $(\gamma_{\ell,v}, M_{\ell,v})$ where $\gamma_{\ell,v} = \delta(\ell, \sigma_{i+1})$ and $M_{\ell,v}$ is the minimal model of $\gamma_{\ell,v}$ with respect to $v + \tau_{i+1} - \tau_i$ chosen in G. We then gather all the triggers in $\mathcal{Q}_i = \bigcup_{(\ell,v) \in K_i} \mathsf{trig}_\varphi(M_{\ell,v}, \gamma_{\ell,v}, v + \tau_{i+1} - \tau_i)$, and let $\rho' = (\sigma_1 \cup \mathcal{Q}_1, \tau_1)(\sigma_2 \cup \mathcal{Q}_2, \tau_2) \cdots$. Then, it can be shown that each component has an accepting run over ρ'. By definition, the generalised Büchi acceptance condition on $\mathcal{C}$ is fulfilled exactly when the Büchi acceptance condition on each of the components is fulfilled. It follows that $\mathcal{C}$ accepts ρ', and hence $\rho \in \mathsf{proj}_{\mathsf{AP}}([\![\mathcal{C}]\!])$. The other direction of the proof consists of building an accepting run G of $\mathcal{A}_\varphi$ over $\mathsf{proj}_{\mathsf{AP}}(\rho')$ from an accepting run of $\mathcal{C}$ over $\rho' \in [\![\mathcal{C}]\!]$. At each level of G, the truth values of the triggers in ρ' are used to guide the construction of minimal models. □

5 Handling full MITL

We can now extend our translation to full MITL, i.e. allowing operators $\mathbf{U}_{[a,b]}$ and $\mathbf{R}_{[a,b]}$ with $0 < a < b < +\infty$. For these two types of operators, we cannot rely on a single clock in the components anymore. For instance, consider the formula $\varphi = \mathbf{G}(p \Rightarrow \chi)$ with $\chi = \mathbf{F}_{[1,2]} q$. Imagine that $\mathcal{A}_\varphi$ reads the prefix

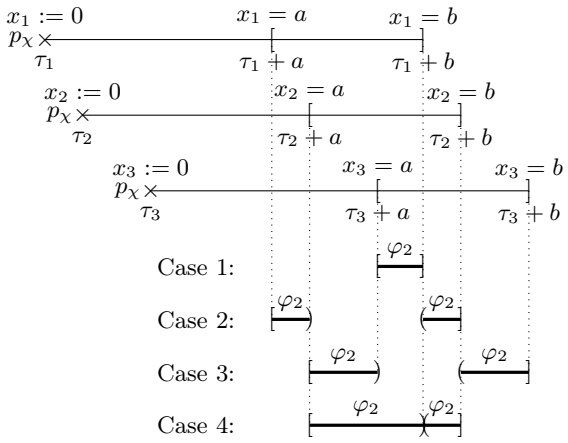

Fig. 7. How to split cases to satisfy the formula $\chi = \varphi_1 \, \mathbf{U}_{[a,b]} \, \varphi_2$.

$(\{p\}, 0)(\{p\}, 0.5)$. At this point, its configuration is $\{(\varphi, 0), (\chi, 0), (\chi, 0.5)\}$. It is not possible, as before, to drop one of the two states in location χ as the following futures can happen: if we read $(\{q\}, 1)$, obligation $(\chi, 0)$ is fulfilled but not $(\chi, 0.5)$; if we read $(\{q\}, 2.5)$ then the obligation $(\chi, 0.5)$ is fuilfilled but not $(\chi, 0)$. Therefore, we must keep track of the two obligations separately. It is, however, not clear how to find an *a priori* bound on the number of clocks. This is the role of the *interval semantics* introduced in [11] for OCATAs resulting from MITL formulae over infinite words. In this interpretation of OCATAs, valuations of the clocks are no longer *points* but *intervals* meant to approximate sets of (singular) valuations: $(\ell, [\alpha, \beta])$ means that there *are* clock copies with valuations α and β in ℓ, yet there *could be* more copies in ℓ with valuations in (α, β). In this semantics, we can *merge* non-deterministically two copies $(\ell, [\alpha_1, \beta_1])$ and $(\ell, [\alpha_2, \beta_2])$ into a single copy $(\ell, [\alpha_1, \beta_2])$ (assuming $\alpha_1 \leq \beta_2$), in order to keep the number of clock copies below a fixed threshold, and thus obtain an equivalent TA. It has been shown in [11] that, for the OCATA $\mathcal{A}_\varphi$, with $\varphi \in$ MITL, the interval semantics is sufficient to retain the language of the formula, with TAs having at most $M(\varphi) = |\varphi| \times \max_{I \in \mathcal{I}_\varphi}(\max(4 \times \lceil \inf(I)/|I| \rceil + 2, 2 \times \lceil \sup(I)/|I| \rceil + 2))$ clocks, where $\mathcal{I}_\varphi$ is the set of intervals that appear in φ: more precisely, each subformula with topmost operator $\mathbf{U}_I$ (respectively, $\mathbf{R}_I$) contributes to $4 \times \lceil \inf(I)/|I| \rceil + 2$ (respectively, $2 \times \lceil \sup(I)/|I| \rceil + 2$) more clocks.

Our solution is twofold in this context: (i) we propose a better approximation by intervals that allows us to cut, up to a factor of two, the number of clock copies we must keep in memory; (ii) instead of a single TA, as in [11], we provide a GBTA, with one component per temporal subformula of φ. The component TA are much more involved than for $\mathsf{MITL}_{0,\infty}$, thus we do not give them explicitly, but rather explain the main ideas.

We start by developing our new merging strategy on an example, to explain how it is different from [11]. Consider $\chi = \varphi_1 \, \mathbf{U}_{[a,b]} \, \varphi_2$ with $0 < a < b < +\infty$ and

the situation depicted in Fig. 7, where the trigger p_χ is pulled at three positions of time stamps τ_1, τ_2, and τ_3. We suppose that φ_1 holds at all three positions. The picture presents four different cases corresponding to the four possible situations where the occurrences of φ_2 fulfil the three pending obligations. Case 1 is when a position in $[\tau_3 + a, \tau_1 + b]$ satisfies φ_2, hence all three obligations are resolved at once. This case can be checked using only clocks x_3 and x_1. In case 2, the first obligation is resolved by an occurrence of φ_2 with time stamp in $[\tau_1 + a, \tau_2 + a)$, while the two others are resolved by an occurrence in $(\tau_1 + b, \tau_2 + b]$. Thus, case 2 can be checked using only clocks x_1 and x_2. Now consider the remaining cases: if no occurrences of φ_2 appear in $[\tau_1 + a, \tau_2 + a) \cup [\tau_3 + a, \tau_1 + b]$, one occurrence of φ_2 must necessarily happen in $[\tau_2 + a, \tau_3 + a)$, while the other should be in $(\tau_1 + b, \tau_3 + b]$, which can only be checked using three clocks x_1, x_2 and x_3. We avoid this by splitting this case into two further cases (cases 3 and 4) that can be checked with only two clocks. Specifically, case 3 can be checked using only clocks x_2 and x_3; and case 4 using only clocks x_2 and x_1.

Observe that these cases can be categorised into two groups: one where φ_2 should occur in a single interval whose endpoints use two distinct clocks (case 1), another where φ_2 should occur in two half-open intervals whose both endpoints use the same two distinct clocks (cases 2, 3 and 4). In each of the two groups, it must be understood how a new obligation (added by pulling the trigger p_χ) modifies the situation. With only one interval, if a new obligation for φ_2 appear as a new interval $[\tau + a, \tau + b]$, either the new obligation is implied by the current one, in which case we are done, or the two intervals intersect and we do a further split (non-deterministically) into cases 1, 3 and 4, or they are disjoint and we keep both intervals in memory. The latter situation cannot happen too often since intervals are non-singular; more precisely, this will happen at most $\lceil(\inf(I)/|I|)+1\rceil$ times ($I = [a, b]$). With two intervals, either the new obligation is already implied by current obligations, or $[\tau + a, \tau + b]$ is not implied by the current obligations and we add this new interval in memory as before (again, this cannot happen more than $\lceil(\inf(I)/|I|) + 1\rceil$ times).

In the end, following the same lines as [11], we can build a component $\mathcal{C}_\chi$ for each subformula $\chi = \varphi_1 \mathbf{U}_{[a,b]} \varphi_2$ with $N(\chi) = 2 \times \lceil(\inf(I)/|I|)+1\rceil+2$ clocks (the two additional clocks are used to deal easily with some special cases), which is roughly half of the previous bound on the number of clocks [11]. In the locations, we can handle the clocks in pairs and use a queue of size $N(\chi)/2$ to keep track of which case we fall into and which clocks are used to represent the endpoints of intervals. It follows that the number of locations is exponential in $N(\chi)$. A similar construction, using $2 \times \lceil(\inf(I)/|I|) + 1\rceil$ clocks, builds a component $\mathcal{C}_\chi$ for each subformula $\chi = \varphi_1 \, \mathbf{R}_{[a,b]} \, \varphi_2$. In this case, we have to consider unions of intervals, which are easier to deal with.

Theorem 7. *For all* MITL *formulae* φ, $\mathsf{proj}_{\mathsf{AP}}([\![\mathcal{C}_{init} \times \prod_{\chi \in \Phi} \mathcal{C}_\chi]\!]) = [\![\varphi]\!]$.

Proof (Sketch.). We follow the same lines of the proof of Proposition 6, i.e. relating the accepting runs of $\mathcal{A}_\varphi$ with the accepting runs of $\mathcal{C} = \mathcal{C}_{init} \times \prod_{\chi \in \Phi} \mathcal{C}_\chi$. To show that $[\![\mathcal{A}_\varphi]\!] \subseteq \mathsf{proj}_{\mathsf{AP}}([\![\mathcal{C}]\!])$, we use the same construction of the Hintikka

sequence over $2^{\mathsf{AP}\cup\mathsf{AP}_\varphi}$. Note that we have the accepting run G, so that we know in advance *how each obligation is to be fulfilled in the future*. In particular, we use this knowledge to resolve the non-determinism in components of the form $\mathcal{C}_{\varphi_1 \mathbf{U}_{[a,b]} \varphi_2}$ or $\mathcal{C}_{\varphi_1 \mathbf{R}_{[a,b]} \varphi_2}$. The other direction is also similar. □

6 Implementation

We have implemented our translation from MITL formulae to generalised Büchi timed automata in a tool called MightyL, written in OCaml. From a formula φ, it produces the GBTA $\mathcal{C}$, described in previous sections, in the XML format used by Uppaal, as well as the generalised Büchi condition written as a very simple LTL formula. When the input formula is in $\mathsf{MITL}_{0,\infty}$, the translation can be done in polynomial time. For the general case, it runs in exponential time (assuming a succinct encoding of constants, as is the case here). We can then use Uppaal [27] to check the satisfiability of φ over finite timed words, or LTSmin [24] with opaal front-end to check satisfiability over infinite timed words. To maximise compatibility with model-checking tools, we use several helper variables in the output XML file, e.g. a Boolean variable for each atomic proposition and a `loc` variable in each component for the current location. The synchronisation is done in a round-robin fashion with a counter variable `N`: initially, `N` is set to 0, allowing the model (to be model-checked) to take a transition and set the truth values of the atomic propositions. Then, `N` loops from 1 to the number of components of $\mathcal{C}$, allowing each component to read the atomic propositions and take a corresponding transition. Finally, `N` is set back to 0 and we start over again. For the finite-word case, this also enables to check that all components have been synchronised properly (`N` $= 0$) while in the final location. Our tool is publicly available, and can even be executed directly on the website

`http://www.ulb.ac.be/di/verif/mightyl`

Compared to the simplified version we studied in this article, MightyL also allows for (semi-)open intervals. Since it can also deal with next and dual-next operators, we can verify formulae like $\neg \mathbf{X}_{[1,2)} p$. All the following tests have been performed on a MacBook Pro 2.7GHz with 8Go RAM.

We check the satisfiability of MITL formulae on examples, inspired by the benchmarks of [11, 19]. For $k \in \mathbb{N}$ and an interval I, we consider the satisfiable formulae: $F(k, I) = \bigwedge_{i=1}^{k} \mathbf{F}_I p_i$, $G(k, I) = \bigwedge_{i=1}^{k} \mathbf{G}_I p_i$, $U(k, I) = (\cdots (p_1 \, \mathbf{U}_I \, p_2) \, \mathbf{U}_I \cdots) \mathbf{U}_I p_k$, $R(k, I) = (\cdots (p_1 \mathbf{R}_I p_2) \mathbf{R}_I \cdots) \mathbf{R}_I p_k$, and $\theta(k, I) = \neg((\bigwedge_{i=1}^{k} \mathbf{GF} p_i) \Rightarrow \mathbf{G}(q \Rightarrow \mathbf{F}_I r))$. We also consider an example inspired from motion planning problems via MITL specifications as in [25, 31]. In this benchmark, a robot must visit some target points $t_1, t_2, t_3, \ldots, t_k$ within given time frames (in our case, t_i must be seen in time frame $[3(i-1), 3i]$), while enforcing a safety condition $\mathbf{G}\neg p$. This specification is modelled by the satisfiable MITL formula $\mu(k) = \bigwedge_{i=1}^{k} \mathbf{F}_{[3(i-1),3i]} t_i \wedge \mathbf{G}\neg p$. In Table 1, we report on the time taken by the execution of MightyL; LTSmin (split into the time taken by opaal front-end to translate the model into C++ code, the compilation time of the resulting

Table 1. Execution time for the satisfiability check of benchmarks of [11, 19]. For LTSMIN, the three columns reported correspond to the translation into C++, the compilation and the actual model-checking, respectively.

Formula	MIGHTYL	LTSMIN	UPPAAL	Formula	MIGHTYL	LTSMIN	UPPAAL
$F(5,[0,\infty))$	9ms	3.48s/2.18s/0.12s	0.75s	$U(5,[0,\infty))$	16ms	1.90s/1.44s/0.05s	0.41s
$F(5,[0,2])$	7ms	3.76s/2.23s/0.15s	0.84s	$U(5,[0,2])$	8ms	2.08s/1.54s/0.06s	0.42s
$F(5,[2,\infty))$	6ms	3.76s/2.26s/0.91s	1.64s	$U(5,[2,\infty))$	8ms	2.08s/1.53s/0.09s	0.52s
$F(3,[1,2])$	70ms	6m5.15s/38.01s/0.22s	9.00s	$U(3,[1,2])$	49ms	4m0.14s/23.54s/0.09s	4.92s
$F(5,[1,2])$	70ms	>15m	2m6s	$U(5,[1,2])$	97ms	>15m	21.80s
$G(5,[0,\infty))$	10ms	3.83s/2.43s/0.05s	0.75s	$R(5,[0,\infty))$	7ms	1.86s/1.42s/0.03s	0.40s
$G(5,[0,2])$	10ms	4.01s/2.51s/0.10s	0.82s	$R(5,[0,2])$	7ms	1.97s/1.44s/0.03s	0.40s
$G(5,[2,\infty))$	9ms	4.06s/2.47s/0.04s	0.85s	$R(5,[2,\infty))$	7ms	1.92s/1.42s/0.03s	0.42s
$G(5,[1,2])$	15ms	7.81s/2.99s/0.09s	1.12s	$R(5,[1,2])$	10ms	5.37s/2.16s/0.04s	0.62s
$\mu(1)$	13ms	-	0.39s	$\theta(1,[100,1000])$	9ms	1.88s/1.74s/0.04s	0.25s
$\mu(2)$	21ms	-	2.33s	$\theta(2,[100,1000])$	13ms	5.04s/3.17s/0.19s	0.86s
$\mu(3)$	76ms	-	15.77s	$\theta(3,[100,1000])$	14ms	36.57s/16.27s/3.20s	21.84s
$\mu(4)$	87ms	-	2m23s	$\theta(4,[100,1000])$	15ms	5m30s/4m18s/2m16s	18m39s

Table 2. Validity and redundancy checking of MITL formulae.

Formula	MIGHTYL	LTSMIN	UPPAAL	[16]
$\mathbf{F}_{[0,30]}(p \Rightarrow \mathbf{G}_{[0,20]}p)$ (tautology)	7ms	0.98s	0.32s	7s
$\mathbf{G}_{[0,30]}\neg p \vee \mathbf{F}_{[0,20]}p$ (valid, i.e. satisfiable and not tautology)	13ms	1.66s	0.30s	not considered
$\mathbf{F}_{[0,30]}p \wedge \mathbf{F}_{[0,20]}p$ (valid but redundant)	24ms	3.39s	0.79s	14s
$\mathbf{G}_{[0,20]}\mathbf{F}_{[0,20]}p \wedge \mathbf{G}_{[0,40]}p \wedge \mathbf{F}_{[20,40]}\top$ (valid but redundant)	60ms	2m58s	4.94s	not considered

C++ code, and the time taken by LTSMIN for the actual model-checking); and UPPAAL, on all these examples (for the motion planning, only finite words are relevant, hence we report only on the UPPAAL running time).

We also report on the benchmarks found in [16], where the debugging of formal specifications of cyber-physical systems is reduced to MITL non-satisfiability. More precisely, we check formulae for *validity* and *redundancy*. In [16], a formula φ is called *valid* (with respect to a specification goal) if φ is neither unsatisfiable nor a tautology, i.e. φ and $\neg\varphi$ are both satisfiable. A conjunct φ_1 of formula $\varphi = \bigwedge_{i=1}^{k} \varphi_i$ is redundant if and only if $\bigwedge_{i=2}^{k} \varphi_i$ implies φ_1. This is true if and only if $\psi = \bigwedge_{i=2}^{k} \varphi_i \Rightarrow \varphi_1$ is valid, i.e. if and only if $\neg\psi$ is not satisfiable. For instance, $\mathbf{F}_{[0,30]}p$ is redundant in $\mathbf{F}_{[0,30]}p \wedge \mathbf{F}_{[0,20]}p$, and $\mathbf{G}_{[0,20]}\mathbf{F}_{[0,20]}p$ is redundant in $\mathbf{G}_{[0,20]}\mathbf{F}_{[0,20]}p \wedge \mathbf{G}_{[0,40]}p \wedge \mathbf{F}_{[20,40]}\top$. We check the validity and redundancy of several formulae considered in [16] and report the results in Table 2. For reference, we copy the execution time reported in [16] for these checks.[8] We also consider some new formulae specific to our pointwise semantics.

Finally, recall that one technical part of the constructions of component Büchi timed automata is the minimal model simplification $\mathsf{mm}(\varphi)$. Our components remain correct if we replace everywhere $\mathsf{mm}(\varphi)$ by φ (i.e. $\widehat{\varphi}$ simply becomes $\overline{\varphi}$). On some instances of the previous benchmarks, the influence on the execution time of the satisfiability checks is tremendous (differences on the execution time of MIGHTYL negligible, since the tool always answers in less than a second). For instance, over $F(5,[0,\infty))$, LTSMIN shows a 17% overhead. For $F(5,[0,2])$,

[8] These numbers are only for reference and should not be taken as a direct comparison since, contrary to us, [16] considers a bounded continuous semantics of MITL.

LTSmin experiences a 5% overhead, while Uppaal has a 12% overhead. For formulae $F(5, [2, \infty))$, $F(3, [1, 2])$, $F(5, [1, 2])$, the situation is even worse since Uppaal stops responding before the timeout of fifteen minutes. LTSmin also hangs on $F(3, [1, 2])$ before the timeout. On the motion planning example, the overhead is also significant for Uppaal, e.g. 80% for μ_2, and, for μ_3 and μ_4, Uppaal does not respond anymore before the timeout. Finally, on the two unsatisfiable examples of the redundancy check, LTSmin and Uppaal have overheads of 70%/3% and 630%/230%, respectively.

7 Conclusion and perspectives

In this work, we proposed a new compositional construction from MITL to timed automata which we implemented the tool MightyL, enabling easy automata-based model-checking of full MITL. For future work, since the structure of the formula is preserved in our construction, we want to investigate antichain-based heuristics to allow more performance boost. For MightyL, we plan to add native support for ECL [33] operators which eases the writing of specifications, as well as past operators and counting operators [23].

Acknowledgements. We thank the reviewers of this article that help us clarify its overall presentation. The third author would like to thank Andreas Engelbredt Dalsgaard, Alfons Laarman and Jeroen Meijer for their technical help with opaal and LTSmin.

References

1. Abid, N., Dal-Zilio, S., Botlan, D.L.: A formal framework to specify and verify real-time properties on critical systems. International Journal of Critical Computer-Based Systems 5(1/2), 4–30 (2014)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM 43(1), 116–146 (1996)
4. Alur, R., Henzinger, T.A.: Real-time logics: Complexity and expressiveness. Information and Computation 104(1), 35–77 (1993)
5. Barnat, J., Brim, L., Havel, V., Havlícek, J., Kriho, J., Lenco, M., Rockai, P., Still, V., Weiser, J.: DivinE 3.0 - an explicit-state model checker for multithreaded C & C++ programs. In: CAV'13. LNCS, vol. 8044, pp. 863–868. Springer (2013)
6. Bartocci, E., Bortolussi, L., Nenzi, L.: A temporal logic approach to modular design of synthetic biological circuits. In: CMSB'13. LNCS, vol. 8130, pp. 164–177. Springer (2013)
7. Bersani, M.M., Rossi, M., San Pietro, P.: A tool for deciding the satisfiability of continuous-time metric temporal logic. Acta Informatica 53(2), 171–206 (2016)
8. Bloem, R., Cimatti, A., Pill, I., Roveri, M.: Symbolic implementation of alternating automata. International Journal of Foundations of Computer Science 18(4), 727–743 (2007)

9. Bouyer, P., Colange, M., Markey, N.: Symbolic optimal reachability in weighted timed automata. In: CAV'16. LNCS, vol. 9779, pp. 513–530. Springer (2016)
10. Brihaye, T., Estiévenart, M., Geeraerts, G.: On MITL and alternating timed automata. In: FORMATS'13. LNCS, vol. 8053, pp. 47–61. Springer (2013)
11. Brihaye, T., Estiévenart, M., Geeraerts, G.: On MITL and alternating timed automata of infinite words. In: FORMATS'14. LNCS, vol. 8711. Springer (2014)
12. Bulychev, P.E., David, A., Larsen, K.G., Li, G.: Efficient controller synthesis for a fragment of $\mathrm{MTL}_{0,\infty}$. Acta Informatica 51(3-4), 165–192 (2014)
13. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV2: An opensource tool for symbolic model checking. In: CAV'02. LNCS, vol. 2404, pp. 359–364. Springer (2002)
14. Claessen, K., Een, N., Sterin, B.: A circuit approach to LTL model checking. In: FMCAD'13. IEEE (2013)
15. De Moura, L., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Communications of the ACM 54(9), 69–77 (Sep 2011)
16. Dokhanchi, A., Hoxha, B., Fainekos, G.: Formal requirement debugging for testing and verification of cyber-physical systems. Research Report 1607.02549, arXiv (2016)
17. D'Souza, D., Matteplackel, R.: A clock-optimal hierarchical monitoring automaton construction for mitl. Research Report 2013-1, IIS (2013), `http://www.csa.iisc.ernet.in/TR/2013/1/lics2013-tr.pdf`
18. Fu, J., Topcu, U.: Computational methods for stochastic control with metric interval temporal logic specifications. In: CDC'15. pp. 7440–7447. IEEE (2015)
19. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: CAV'01. LNCS, vol. 2102, pp. 53–65. Springer (2001)
20. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: PSTV'95. pp. 3–18. Chapman & Hall (1995)
21. Hammer, M., Knapp, A., Merz, S.: Truly on-the-fly LTL model checking. In: TACAS'05. LNCS, vol. 3440, pp. 191–205. Springer (2005)
22. Hirshfeld, Y., Rabinovich, A.M.: Logics for real time: Decidability and complexity. Fundamenta Informaticae 62(1), 1–28 (2004)
23. Hirshfeld, Y., Rabinovich, A.M.: An expressive temporal logic for real time. In: MFCS'06. LNCS, vol. 4162, pp. 492–504. Springer (2006)
24. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: High-performance language-independent model checking. In: TACAS'15. LNCS, vol. 9035, pp. 692–707. Springer (2015)
25. Karaman, S.: Optimal Planning with Temporal Logic Specifications. Master's thesis, Massachussetts Institute of Technology (2009)
26. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. In: ISTCS'97. pp. 147–158. IEEE (1997)
27. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)
28. Maler, O., Nickovic, D., Pnueli, A.: From MITL to timed automata. In: FORMATS'06. LNCS, vol. 4202, pp. 274–289. Springer (2006)
29. Muller, D.E., Saoudi, A., Schupp, P.E.: Alternating automata, the weak monadic theory of the tree, and its complexity. In: ICALP'86. LNCS, vol. 226, pp. 275–283. Springer (1986)
30. Ouaknine, J., Worrell, J.: On the decidability and complexity of metric temporal logic over finite words. Logical Methods in Computer Science 3(1) (2007)
31. Plaku, E., Karaman, S.: Motion planning with temporal-logic specifications: Progress and challenges. AI Communications 29, 151–162 (2016)

32. Pnueli, A.: The temporal logic of programs. In: FOCS'77. pp. 46–57. IEEE (1977)
33. Raskin, J.F., Schobbens, P.Y.: The logic of event clocks: Decidability, complexity and expressiveness. Journal of Automata, Languages and Combinatorics 4(3), 247–282 (1999)
34. Rozier, K.Y., Vardi, M.Y.: A multi-encoding approach for LTL symbolic satisfiability checking. In: FM'11. LNCS, vol. 6664, pp. 417–431. Springer (2011)
35. Thierry-Mieg, Y.: Symbolic model-checking using ITS-tools. In: TACAS'15. LNCS, vol. 9035, pp. 231–237. Springer (2015)
36. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Logics for Concurrency, LNCS, vol. 1043, pp. 238–266. Springer (1996)
37. Wilke, T.: Specifying timed state sequences in powerful decidable logics and timed automata. In: FTRTFT'94. LNCS, vol. 863, pp. 694–715. Springer (1994)
38. Wulf, M.D., Doyen, L., Maquet, N., Raskin, J.F.: Antichains: Alternative algorithms for LTL satisfiability and model-checking. In: TACAS'08. LNCS, vol. 4963, pp. 63–77. Springer (2008)
39. Zhou, Y., Maity, D., Baras, J.S.: Timed automata approach for motion planning using metric interval temporal logic. Research Report 1603.08246, arXiv (2016)