Real-time Synthesis is Hard!

Thomas Brihaye, Morgane Estiévenart, Gilles Geeraerts, Hsi-Ming Ho, Benjamin Monmege, Nathalie Sznajder

Real-time Synthesis is Hard!*

Thomas Brihaye[1], Morgane Estiévenart[1], Gilles Geeraerts[2], Hsi-Ming Ho[1], Benjamin Monmege[3], Nathalie Sznajder[4]

[1] Université de Mons, Belgium,
thomas.brihaye,morgane.estievenart,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@lif.univ-mrs.fr
[4] Sorbonne Universités, UPMC, LIP6, France, nathalie.sznajder@lip6.fr

Abstract. We study the reactive synthesis problem (RS) for specifications given in Metric Interval Temporal Logic (MITL). RS is known to be undecidable in a very general setting, but on infinite words only; and only the very restrictive BResRS subcase is known to be decidable (see D'Souza *et al.* and Bouyer *et al.*). In this paper, we precise the decidability border of MITL synthesis. We show RS is undecidable on finite words too, and present a landscape of restrictions (both on the logic and on the possible controllers) that are still undecidable. On the positive side, we revisit BResRS and introduce an efficient on-the-fly algorithm to solve it.

1 Introduction

The design of programs that respect real-time specifications is a difficult problem with recent and promising advances. Such programs must handle thin timing behaviours, are prone to errors, and difficult to correct a posteriori. Therefore, one road to the design of correct real-time software is the use of automatic synthesis methods, that *build*, from a specification, a program which is correct by construction. To this end, *timed games* are nowadays recognised as the key foundational model for the synthesis of real-time programs. These games are played between a *controller* and an *environment*, that propose actions in the system, modelled as a *plant*. The *reactive synthesis problem* (RS) consists, given a real-time specification, in deciding whether the controller has a winning strategy ensuring that every execution of the plant consistent with this strategy (i.e., no matter the choices of the environment) satisfies the specification. As an example, consider a lift for which we want to design a software verifying certain safety conditions. In this case, the plant is a (timed) automaton, whose states record the current status of the lift (its floor, if it is moving, the button on which users have pushed...), as well as timing information regarding the evolution in-between the different states. On the other hand, the specification is usually given

* More technical details and proofs can be found in the full version of this paper [8]. This work has been supported by The European Union Seventh Framework Programme under Grant Agreement 601148 (Cassting) and by the FRS/F.N.R.S. PDR grant SyVeRLo.

using some real-time logic: in this work, we consider mainly specifications given by a formula of MITL [2], a real-time extension of LTL. Some actions in the plant are controllable (closing the doors, moving the cart), while others belong to the environment (buttons pushed by users, exact timing of various actions inside intervals, failures...). Then, the RS problem asks to compute a controller that performs controllable actions at the right moments, so that, for all behaviours of the environment, the lift runs correctly.

In the *untimed case*, many positive theoretical and practical results have been achieved regarding RS: for instance, when the specification is given as an LTL formula, we know that if a winning strategy exists, then there is one that can be described by a finite state machine [20]; and efficient LTL synthesis algorithms have been implemented [15,3]. Unfortunately, in the real-time setting, the picture is not so clear. Indeed, a winning strategy in a timed game might need unbounded memory to recall the full prefix of the game, which makes the real-time synthesis problem a *hard* one. This is witnessed by three papers presenting negative results: D'Souza and Madhusudan [13] and Bouyer *et al.* [4] show that RS is undecidable (on finite and infinite words) when the specification is respectively a timed automaton and an MTL formula (the two most expressive formalisms in Fig. 1). More recently, Doyen *et al.* show [12] that RS is undecidable for MITL specifications over infinite words; but leave the finite words case open.

When facing an undecidability result, one natural research direction consists in considering subcases in order to recover decidability: here, this amounts to considering fragments of the logic, or restrictions on the possible controllers. Such results can also be found in the aforementioned works. In [13], the authors consider a variant of RS, called *bounded resources reactive synthesis* (BResRS) where the number of clocks and the set of guards that the controller can use are fixed a priori, and the specification is given by means of a timed automaton. By coupling this technique with the translation of MITL into timed automata [7], one obtains a 3-EXPTIME procedure (in the finite and infinite words cases). Unfortunately, due to the high cost of translating MITL into timed automata and the need to construct its entire deterministic region automaton, this algorithm is unlikely to be amenable to implementation. Then, [4] presents an on-the-fly algorithm for BResRS with MTL specifications (MTL is a strict superset of MITL), on finite words, but their procedure runs in non-primitive recursive time.

Hence, the decidability status of the synthesis problem (with MITL requirements) still raises several questions, namely: (*i*) Can we relax the restrictions in the definition of BResRS while retaining decidability? (*ii*) Is RS decidable on finite words, as raised in [12]? (*iii*) Are there meaningful restrictions of the logic that make RS decidable? (*iv*) Can we devise an on-the-fly, efficient, algorithm that solves BResRS in 3-EXPTIME as in [13]? In the present paper, we provide answers to those questions.First, we consider the additional IRS, BPrecRS and BClockRS problems, that introduce different levels of restrictions. IRS requests the controller to be a timed automaton. BPrecRS and BClockRS are further restrictions of IRS where respectively the set of guards and the set of clocks of the controller are fixed a priori. Thus, we consider the following hierarchy of

problems: $\mathsf{RS} \supseteq \mathsf{IRS} \supseteq \begin{matrix}\mathsf{BPrecRS}\\ \mathsf{BClockRS}\end{matrix} \supseteq \mathsf{BResRS}$. Unfortunately, while IRS, BPrecRS and BClockRS seem to make sense in practice, they turn out to be undecidable both on finite and infinite words—an answer to points (*i*) and (*ii*). Our proofs are based on a *novel* encoding of halting problem for deterministic channel machines. By contrast, the undecidability results of [4] (for MTL) are reductions from the same problem, but their encoding relies heavily on the ability of MTL to express *punctual constraints* like 'every a event is followed by a b event *exactly* one time unit later', which is not allowed by MITL. To the best of our knowledge, our proofs are the first to perform such a reduction in a formalism that disallows punctual requirements—a somewhat unexpected result. Then, we answer point (*iii*) by considering a hierarchy of syntactic subsets of MITL (see Fig. 1) and showing that, for all these subsets, BPrecRS and BClockRS (hence also IRS and RS) remain undecidable, on finite and infinite words. Note that the undecidability proof of [13] cannot easily be adapted to cope with these cases, because it needs a mix of open and closed constraints; while we prove undecidable very weak fragments of MITL where only closed or only open constraints are allowed. All these negative results shape a precise picture of the decidability border for real-time synthesis (in particular, they answer open questions from [4],[9] and [12]). On the positive side, we answer point (*iv*) by devising an on-the-fly algorithm to solve BResRS (in the finite words case) that runs in 3-EXPTIME. It relies on one-clock alternating timed automata (as in [4], but unlike [13] that use timed automata), and on the recently introduced *interval semantics* [7].

2 Reactive synthesis of timed properties

Let Σ be a finite alphabet. A (finite) timed word[5] over Σ is a finite word $\sigma = (\sigma_1, \tau_1) \cdots (\sigma_n, \tau_n)$ over $\Sigma \times \mathbb{R}^+$ with $(\tau_i)_{1 \leqslant i \leqslant n}$ a non-decreasing sequence of non-negative real numbers. We denote by $T\Sigma^\star$ the set of finite timed words over Σ. A *timed language* is a subset L of $T\Sigma^\star$.

Timed logics. We consider the reactive synthesis problem against various real-time logics, all of them being restrictions of Metric Temporal Logic (MTL) [16]. The logic MTL is a timed extension of LTL, where the temporal modalities are labelled with a timed interval. The formal syntax of MTL is given as follows:

$$\varphi := \top \mid a \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi \,\mathsf{U}_I\, \varphi$$

where $a \in \Sigma$ and I is an interval over $\mathbb{R}^+$ with endpoints in $\mathbb{N} \cup \{+\infty\}$.

We consider the *pointwise semantics* and interpret MTL formulas over timed words. The semantics of a formula φ in MTL is defined inductively in the usual way. We recall only the semantics of U: given $\sigma = (\sigma_1, \tau_1) \cdots (\sigma_n, \tau_n) \in T\Sigma^\star$, and a position $1 \leqslant i \leqslant n$, we let $(\sigma, i) \models \varphi_1 \,\mathsf{U}_I\, \varphi_2$ if there exists $j > i$ such that $(\sigma, j) \models \varphi_2$, $\tau_j - \tau_i \in I$, and $(\sigma, k) \models \varphi_1$, for all $i < k < j$.

[5] In order to keep the discussion focused and concise, we give the formal definitions for finite words only. It is straightforward to adapt them to the infinite words case.

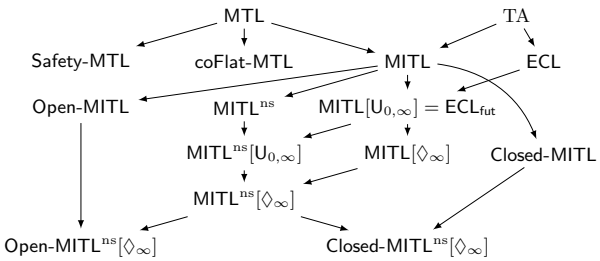

Fig. 1. All the fragments of MITL for which BPrecRS and BClockRS are undecidable (hence also RS and IRS). $A \to B$ means that A strictly contains B.

With $\bot := \neg\top$, we can recover the 'next' operator $\bigcirc_I \varphi := \bot \,\mathsf{U}_I\, \varphi$, and we rely on the usual shortcuts for the 'finally', 'globally' and 'dual-until' operators: $\Diamond_I \varphi := \top \,\mathsf{U}_I\, \varphi$, $\Box_I \varphi := \neg \Diamond_I \neg\varphi$ and $\varphi_1 \,\widetilde{\mathsf{U}}_I\, \varphi_2 := \neg((\neg\varphi_1)\,\mathsf{U}_I\,(\neg\varphi_2))$. We also use the non-strict version of the 'until' operator $\varphi_1 \,\overline{\mathsf{U}}_I\, \varphi_2$, defined as $\varphi_2 \vee (\varphi_1 \wedge \varphi_1 \,\mathsf{U}_I\, \varphi_2)$ (if $0 \in I$) or $\varphi_1 \wedge \varphi_1 \,\mathsf{U}_I\, \varphi_2$ (if $0 \notin I$). This notation yields the corresponding non-strict operators $\overline{\Diamond}\,\varphi$ and $\overline{\Box}\,\varphi$ in the natural way. When the interval I is the entire set of the non-negative real numbers, the subscript is often omitted. We say that σ satisfies the formula φ, written $\sigma \models \varphi$ if $(\sigma, 1) \models \varphi$, and we denote by $\mathcal{L}(\varphi)$ the set of all timed words σ such that $\sigma \models \varphi$.

We consider mainly a restriction of MTL called MITL (for Metric Interval Temporal Logic), in which the intervals are restricted to non-singular ones. We denote by Open-MITL the open fragment of MITL: in negation normal form, each subformula $\varphi_1 \,\mathsf{U}_I\, \varphi_2$ has either I open or $\inf(I) = 0$ and I right-open, and each subformula $\varphi_1 \,\widetilde{\mathsf{U}}_I\, \varphi_2$ has I closed. Then, a formula is in Closed-MITL if it is the negation of an Open-MITL formula. By [7], Open-MITL formulas (respectively, Closed-MITL formulas) translate to open (closed) timed automata [17], i.e., all clock constraints are strict (non-strict). Two other important fragments of MTL considered in the literature consist of Safety-MTL [19], where each subformula $\varphi_1 \,\mathsf{U}_I\, \varphi_2$ has I bounded in negation normal form, and coFlat-MTL [5], where the formula satisfies the following in negation normal form: (i) in each subformula $\varphi_1 \,\mathsf{U}_I\, \varphi_2$, if I is unbounded then $\varphi_2 \in \mathsf{LTL}$; and (ii) in each subformula $\varphi_1 \,\widetilde{\mathsf{U}}_I\, \varphi_2$, if I is unbounded then $\varphi_1 \in \mathsf{LTL}$.

For all of these logics L, we can consider several restrictions. The restriction in which only the non-strict variants of the operators ($\overline{\Diamond}$, $\overline{\Box}$, *etc.*) are allowed is denoted by L^{ns}. The fragment in which all the intervals used in the formula are either unbounded, or have a left endpoint equal to 0 is denoted by $\mathsf{L}[\mathsf{U}_{0,\infty}]$. In this case, the interval I can be replaced by an expression of the form $\sim c$, with $c \in \mathbb{N}$, and $\sim \in \{<, >, \leqslant, \geqslant\}$. It is known that $\mathsf{MITL}[\mathsf{U}_{0,\infty}]$ is expressively equivalent to $\mathsf{ECL}_{\mathsf{fut}}$ [21], which is itself a syntactic fragment of Event-Clock Logic (ECL). Finally, $\mathsf{L}[\Diamond_\infty]$ stands for the logic where 'until' operators only appear in the form of $\Diamond_I$ or $\Box_I$ with intervals I of the shape $[a, \infty)$ or (a, ∞).

Symbolic transition systems. Let X be a finite set of 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 $\bowtie \in \{<, \leqslant, =, \geqslant, >\}$, $x \in X$ and $c \in \mathbb{Q}^+$. A *valuation* over X is a mapping $\nu\colon X \to \mathbb{R}^+$. The satisfaction of a constraint g by a valuation ν is defined in the usual way and noted $\nu \models g$, and $[\![g]\!]$ is the set of valuations ν satisfying g. For $t \in \mathbb{R}^+$, we let $\nu+t$ be the valuation defined by $(\nu+t)(x) = \nu(x)+t$ for all $x \in X$. For $R \subseteq X$, we let $\nu[R \leftarrow 0]$ be the valuation defined by $(\nu[R \leftarrow 0])(x) = 0$ if $x \in R$, and $(\nu[R \leftarrow 0])(x) = \nu(x)$ otherwise.

Following the terminology of [13,4], a *granularity* is a triple $\mu = (X, m, K)$ where X is a finite set of clocks, $m \in \mathbb{N} \setminus \{0\}$, and $K \in \mathbb{N}$. A constraint g is μ-granular if $g \in \mathcal{G}(X)$ and each constant in g is of the form $\frac{\alpha}{m}$ with an integer $\alpha \leqslant K$. A *symbolic alphabet* Γ based on (Σ, X) is a finite subset of $\Sigma \times \mathcal{G}^{\mathrm{atom}}_{m,K}(X) \times 2^X$, where $\mathcal{G}^{\mathrm{atom}}_{m,K}(X)$ denotes all atomic (X, m, K)-granular clock constraints (i.e., clock constraints g such that $[\![g]\!] = [\![g']\!]$ or $[\![g]\!] \cap [\![g']\!] = \emptyset$, for every (X, m, K)-granular clock constraint g'). Such a symbolic alphabet Γ is said μ-granular. A *symbolic word* $\gamma = (\sigma_1, g_1, R_1) \cdots (\sigma_n, g_n, R_n)$ over Γ generates a set of timed words over Σ, denoted by $tw(\gamma)$ such that $\sigma \in tw(\gamma)$ if $\sigma = (\sigma_1, \tau_1) \cdots (\sigma_n, \tau_n)$, and there is a sequence $(\nu_i)_{0 \leqslant i \leqslant n}$ of valuations with ν_0 the zero valuation, and for all $1 \leqslant i \leqslant n$, $\nu_{i-1} + \tau_i - \tau_{i-1} \models g_i$ and $\nu_i = (\nu_{i-1} + \tau_i - \tau_{i-1})[R_i \leftarrow 0]$ (assuming $\tau_0 = 0$). Intuitively, each (σ_i, g_i, R_i) means that action σ_i is performed, with guard g_i satisfied and clocks in R_i reset.

A *symbolic transition system* (STS) over a symbolic alphabet Γ based on (Σ, X) is a tuple $\mathcal{T} = (S, s_0, \Delta, S_f)$ where S is a possibly infinite set of locations, $s_0 \in S$ is the initial location, $\Delta \subseteq S \times \Gamma \times S$ is the transition relation, and $S_f \subseteq S$ is a set of accepting locations (omitted if all locations are accepting). An STS with finitely many locations is a *timed automaton* (TA) [1]. For a finite path $\pi = s_1 \xrightarrow{b_1} s_2 \xrightarrow{b_2} \cdots \xrightarrow{b_n} s_{n+1}$ of $\mathcal{T}$ (i.e., such that $(s_i, b_i, s_{i+1}) \in \Delta$ for all $1 \leqslant i \leqslant n$), the *trace* of π is the word $b_1 b_2 \cdots b_n$, and π is *accepting* if $s_{n+1} \in S_f$. We denote by $\mathcal{L}(\mathcal{T})$ the language of $\mathcal{T}$, defined as the timed words associated to symbolic words that are traces of finite accepting paths starting in s_0. We say that a timed action $(t, \sigma) \in \mathbb{R}^+ \times \Sigma$ is *enabled* in $\mathcal{T}$ at a pair (s, ν), denoted by $(t, \sigma) \in \mathsf{En}_{\mathcal{T}}(s, \nu)$, if there exists a transition $(s, (\sigma, g, R), s') \in \delta$ such that $\nu + t \models g$. The STS $\mathcal{T}$ is *time-deterministic* if there are no distinct transitions $(s, (\sigma, g_1, R_1), s_1)$ and $(s, (\sigma, g_2, R_2), s_2)$ in Δ and no valuation ν such that $\nu \models g_1$ and $\nu \models g_2$. In a time-deterministic STS $\mathcal{T} = (S, s_0, \delta, S_f)$, for all timed words σ, there is at most one path π whose trace γ verifies $\sigma \in tw(\gamma)$. In that case, we denote by $\delta(s_0, \sigma)$ the unique (if it exists) pair (s, ν) (where $s \in S$ and ν is a valuation) reached after reading $\sigma \in tw(\gamma)$.

Example 1. A time-deterministic TA $\mathcal{P}$ with a single clock x is depicted in Fig. 2. Intuitively, it accepts all timed words σ of the form $w_1 w_2 \cdots w_n$ where each w_i is a timed word such that (*i*) either $w_i = (b, \tau)$; (*ii*) or w_i is a sequence of a's (starting at time stamp τ) of duration at most 1; and w_{i+1} is either of the form (b, τ'), or of the form (a, τ') with $\tau' - \tau > 1$.

Reactive synthesis with plant. To define our reactive synthesis problems, we partition the alphabet Σ into controllable and environment actions Σ_C and Σ_E. Following [13,4], the system is modelled by a time-deterministic TA $\mathcal{P} =$

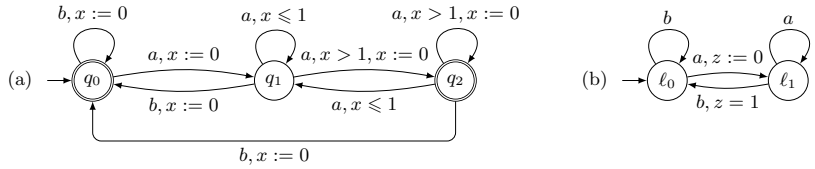

Fig. 2. (a) A time-deterministic STS $\mathcal{P}$ with $X = \{x\}$. Instead of depicting a transition per letter (a, g, R) (with g atomic), we merge several transitions; e.g., we skip the guard, when all the possible guards are admitted. $x := 0$ denotes the reset of x. (b) A time-deterministic STS $\mathcal{T}$. It is a controller to realise $\varphi = \Box(a \Rightarrow \Diamond_{\leqslant 1} b)$ with plant $\mathcal{P}$.

$(Q, q_0, \delta_{\mathcal{P}}, Q_f)$, called the *plant*[6]. Observe that the plant has accepting locations: only those runs ending in a final location of the plant will be checked against the specification. We start by recalling the definition of the general *reactive synthesis* family of problems (RS) [11,12]. It consists in a game played by the controller and the environment, that interact to create a timed word as follows. We start with the empty timed word, and then, at each round, the controller and the environment propose timed actions to be performed by the system—therefore, they must be firable in the plant $\mathcal{P}$—respectively (t, a) and (t', b), with $t, t' \in \mathbb{R}^+$, $a \in \Sigma_C$ and $b \in \Sigma_E$. The timed action with the shortest[7] delay (or the environment action if the controller decides not to propose any action) is performed, and added to the current play for the next round. If both players propose the same delay, we resolve the time non-deterministically.

On those games, we consider a parameterised family of reactive synthesis problems denoted $\mathsf{RS}_s^b(\mathcal{F})$, where $s \in \{u, d\}$; $b \in \{\star, \omega\}$; and $\mathcal{F}$ is one of the formalisms in Fig. 1. An instance of $\mathsf{RS}_s^b(\mathcal{F})$ is given by a specification $S \in \mathcal{F}$ and a plant $\mathcal{P}$, which are interpreted over finite words when $b = \star$ and infinite words when $b = \omega$. The timed language $\mathcal{L}(S)$ is a specification of desired behaviours when $s = d$ and undesired behaviours when $s = u$. Then, $\mathsf{RS}_s^b(\mathcal{F})$ asks whether there exists a strategy for the controller such that all the words in the outcome of this strategy are in $\mathcal{L}(S)$ (or outside $\mathcal{L}(S)$) when we consider desired (or undesired) behaviours (when $s = \omega$, the definition of $\mathcal{L}(S)$ must be the infinite words one). If this is the case, we say that S is *(finite-word) realisable* for the problem under study. For example, $\mathsf{RS}_u^\omega(\mathsf{MITL})$ is the reactive synthesis problem where the inputs are a formula of MITL and a plant, which are interpreted over the infinite words semantics, and where the MITL formula specifies the behaviours that the controller should avoid. Unfortunately, the variants RS are too general, and a winning strategy might require unbounded memory:

Example 2. Consider the alphabet $\Sigma = \Sigma_C \uplus \Sigma_E$ with $\Sigma_C = \{b\}$ and $\Sigma_E = \{a\}$, a plant $\mathcal{P}$ accepting $T\Sigma^\star$, and the specification defined by the MTL formula

[6] We assume that for every location q and every valuation ν, there exists a timed action $(t, \sigma) \in \mathbb{R}^+ \times \Sigma$ and a transition $(q, (\sigma, g, R), q') \in \delta_{\mathcal{P}}$ such that $\nu + t \models g$.

[7] Observe that this is different from [13,4], where the environment can always prevent the controller from playing, even by proposing a longer delay. We claim our definition is more reasonable in practice but all proofs can be adapted to both definitions.

$\varphi = \Box\big((a \wedge \Diamond_{\geqslant 1} a) \Rightarrow \Diamond_{=1} b\big)$. Clearly, a winning strategy for the controller is to remember the time stamps $\tau_1, \tau_2, \ldots$ of all a's, and always propose to play action b one time unit later (note that if the environment blocks the time to prevent the controller from playing its b, the controller wins). However this requires to memorise an unbounded number of time stamps with a great precision.

Restrictions on the RS problem. In practice, it makes more sense to restrict the winning strategy of the controller to be implementable by an STS, which has finitely many clocks (and if possible finitely many locations). Let us define formally what it means for an STS $\mathcal{T} = (S, s_0, \delta)$ to control a plant $\mathcal{P}$. We let $T\Sigma^{\star}_{\mathcal{T},\mathcal{P}}$ be the *set of timed words consistent with* $\mathcal{T}$ *and* $\mathcal{P}$, defined as the smallest set containing the empty timed word, and closed by the following operations. Let σ be a word in $T\Sigma^{\star}_{\mathcal{T},\mathcal{P}}$, with $(q, \nu_{\mathcal{P}}) = \delta_{\mathcal{P}}(q_0, \sigma)$, $T = 0$ if $\sigma = \varepsilon$, and $(c, T) \in \Sigma \times \mathbb{R}^+$ be the last letter of σ otherwise. Then, we extend σ as follows:

- either the controller proposes to play a controllable action (t, b), because it corresponds to a transition that is firable both in the controller and the plant. This action can be played (σ is extended by $(b, T+t)$), as well as any environment action (t', a) with $t' \leqslant t$ (the environment can overtake the controller). Formally, if $\delta(s_0, \sigma) = (s, \nu)$ is defined and $\mathsf{En}_{\mathcal{T}}(s, \nu) \cap \mathsf{En}_{\mathcal{P}}(q, \nu_{\mathcal{P}}) \cap (\mathbb{R}^+ \times \Sigma_C) \neq \emptyset$: for all $(t, b) \in \mathsf{En}_{\mathcal{T}}(s, \nu) \cap \mathsf{En}_{\mathcal{P}}(q, \nu_{\mathcal{P}}) \cap (\mathbb{R}^+ \times \Sigma_C)$, we let $\sigma \cdot (b, T+t) \in T\Sigma^{\star}_{\mathcal{T},\mathcal{P}}$ and $\sigma \cdot (a, T+t') \in T\Sigma^{\star}_{\mathcal{T},\mathcal{P}}$ for all $t' \leqslant t$ and $a \in \Sigma_E$ such that $(t', a) \in \mathsf{En}_{\mathcal{P}}(q, \nu_{\mathcal{P}})$.
- Or the controller proposes nothing, then the environment can play all its enabled actions. Formally, if $\delta(s_0, \sigma) = (s, \nu)$ is defined and $\mathsf{En}_{\mathcal{T}}(s, \nu) \cap \mathsf{En}_{\mathcal{P}}(q, \nu_{\mathcal{P}}) \cap (\mathbb{R}^+ \times \Sigma_C) = \emptyset$ and $\mathsf{En}_{\mathcal{P}}(q, \nu_{\mathcal{P}}) \cap (\mathbb{R}^+ \times \Sigma_E) \neq \emptyset$, we let $\sigma \cdot (a, T+t') \in T\Sigma^{\star}_{\mathcal{T},\mathcal{P}}$ for all $(t', a) \in \mathsf{En}_{\mathcal{P}}(q, \nu_{\mathcal{P}}) \cap (\mathbb{R}^+ \times \Sigma_E)$.
- Otherwise, we declare that every possible future allowed by the plant is valid, i.e., we let $\sigma \cdot \sigma' \in T\Sigma^{\star}_{\mathcal{T},\mathcal{P}}$ for all $\sigma \cdot \sigma' \in \mathcal{L}(\mathcal{P})$. This happens when the controller proposes only actions that are not permitted by the plant while the environment has no enabled actions; or when the controller lost track of a move of the environment during the past.

Then, the MTL *implementable reactive synthesis* problem $\mathsf{IRS}^{\star}_{d}(\mathsf{MTL})$ (on finite words and with desired behaviours) is to decide, given a plant $\mathcal{P}$ and a specification given as an MTL formula φ, whether there exists a set of clocks X, a symbolic alphabet Γ based on (Σ, X), and a time-deterministic STS $\mathcal{T}$ over Γ such that $T\Sigma^{\star}_{\mathcal{T},\mathcal{P}} \cap \mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\varphi) \cup \{\varepsilon\}$.[8]

While the definition of $\mathsf{IRS}^{\star}_{d}(\mathsf{MTL})$ is more practical than that of $\mathsf{RS}^{\star}_{d}(\mathsf{MTL})$, it might still be too general because the clocks and symbolic alphabet the controller can use are not fixed *a priori*. In the spirit of [13,4], we define three variants of IRS. First, the MTL *bounded-resources synthesis problem* $\mathsf{BResRS}^{\star}_{d}(\mathsf{MTL})$ is a restriction of $\mathsf{IRS}^{\star}_{d}(\mathsf{MTL})$ where the granularity of the controller is fixed: given an MTL formula φ, and a granularity $\mu = (X, m, K)$, it asks whether there exists a μ-granular symbolic alphabet Γ based on (Σ, X), and a time-deterministic STS $\mathcal{T}$ over Γ such that $T\Sigma^{\star}_{\mathcal{T},\mathcal{P}} \cap \mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\varphi) \cup \{\varepsilon\}$. Second, the

[8] Empty word ε is added for convenience, in case it is not already in $\mathcal{L}(\varphi)$.

less restrictive MTL *bounded-precision synthesis problem* $\mathsf{BPrecRS}^{\star}_{d}(\mathsf{MTL})$ and MTL *bounded-clocks synthesis problem* $\mathsf{BClockRS}^{\star}_{d}(\mathsf{MTL})$ are the variants of IRS where *only* the precision and *only* the number of clocks are fixed, respectively. Formally, $\mathsf{BPrecRS}^{\star}_{d}(\mathsf{MTL})$ asks, given an MTL formula φ, $m \in \mathbb{N}$, and $K \in \mathbb{N}\backslash\{0\}$, whether there are a finite set X of clocks, an (X, m, K)-granular symbolic alphabet Γ based on (Σ, X), and a time-deterministic STS $\mathcal{T}$ over Γ such that $T\Sigma^{\star}_{\mathcal{T},\mathcal{P}} \cap \mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\varphi) \cup \{\varepsilon\}$. $\mathsf{BClockRS}^{\star}_{d}(\mathsf{MTL})$ is defined similarly with an MTL formula φ, and a finite set of clocks X (instead of m, K) as input.

While we have defined IRS, BPrecRS, BClockRS and BResRS for MTL requirements, and in the finite words, desired behaviours case only, these definitions extend to all the other cases we have considered for RS: infinite words, undesired behaviours, and all fragments of MTL. We rely on the same notations as for RS, writing for instance $\mathsf{BPrecRS}^{\star}_{u}(\mathsf{MITL})$ or $\mathsf{BClockRS}^{\omega}_{d}(\mathsf{coFlat\text{-}MTL})$, etc.

Example 3. Consider the instance of $\mathsf{IRS}^{\star}_{d}(\mathsf{MITL})$ where the plant accepts $T\Sigma^{\star}$ and the specification is $\varphi = \Box(a \Rightarrow \Diamond_{\leqslant 1} b))$. This instance is negative (φ is not realisable), since, for every time-deterministic STS $\mathcal{T}$, $(a, 0) \in T\Sigma^{\star}_{\mathcal{T},\mathcal{P}}$ but is not in $\mathcal{L}(\varphi)$. However, if we consider now the plant $\mathcal{P}$ in Fig. 2(a), we claim that the STS $\mathcal{T}$ with one clock z depicted in Fig. 2(b) realises φ. Indeed, this controller resets its clock z each time it sees the first a in a sequence of a's, and proposes to play a b when z has value 1, which ensures that all a's read so far are followed by a b within 1 time unit. The restrictions enforced by the plant (which can be regarded as a sort of fairness condition) ensure that this is sufficient to realise φ for $\mathsf{IRS}^{\star}_{d}(\mathsf{MITL})$. This also means that φ is realisable for $\mathsf{BPrecRS}^{\star}_{d}(\mathsf{MITL})$ with precision $m = 1$ and $K = 1$; for $\mathsf{BClockRS}^{\star}_{d}(\mathsf{MITL})$ with set of clocks $X = \{z\}$; and for $\mathsf{BResRS}^{\star}_{d}(\mathsf{MITL})$ with granularity $\mu = (\{z\}, 1, 1)$.

3 BPrecRS and BClockRS are undecidable

Let us show that all the variants of BPrecRS and BClockRS are undecidable, whatever formalism from Fig. 1 we consider for the specification. This entails that all variants of RS and IRS are undecidable too (in particular $\mathsf{RS}^{\star}_{d}(\mathsf{ECL})$ which settles an open question of [12] negatively). To this aim, we show undecidability on the weakest formalisms in Fig. 1, namely: coFlat-MTL, Safety-MTL, $\mathsf{Open\text{-}MITL}^{\mathrm{ns}}[\Diamond_{\infty}]$ and $\mathsf{Closed\text{-}MITL}^{\mathrm{ns}}[\Diamond_{\infty}]$. Similar results have been shown for MTL (and for Safety-MTL as desired specifications) in [4] via a reduction from the halting problem for deterministic channel machines, but their proof depends crucially on *punctual* formulas of the form $\overline{\Box}(a \Rightarrow \overline{\Diamond}_{=1} b)$ which are not expressible in MITL. Our original contribution here is to adapt these ideas to a formalism without punctual constraints, which is non-trivial.

Deterministic channel machines. A *deterministic channel machine* (DCM) $\mathcal{S} = \langle S, s_0, s_{halt}, M, \Delta \rangle$ can be seen as a finite automaton equipped with an unbounded fifo channel, where S is a finite set of states, s_0 is the initial state, s_{halt} is the halting state, M is a finite set of messages and $\Delta \subseteq S \times \{m!, m? \mid m \in M\} \times S$ is the transition relation satisfying the following *determinism* hypothesis:

(i) $(s, a, s') \in \Delta$ and $(s, a, s'') \in \Delta$ implies $s' = s''$; (ii) if $(s, m!, s') \in \Delta$ then it is the only outgoing transition from s.

The semantics is described by a graph $G(\mathcal{S})$ with nodes labelled by (s, x) where $s \in S$ and $x \in M^\star$ is the channel content. The edges in $G(\mathcal{S})$ are defined as follows: (i) $(s, x) \xrightarrow{m!} (s', xm)$ if $(s, m!, s') \in \Delta$; and (ii) $(s, mx) \xrightarrow{m?} (s', x)$ if $(s, m?, s') \in \Delta$. Intuitively, these correspond to messages being *written to* or *read from* the channel. A *computation* of $\mathcal{S}$ is then a path in $G(\mathcal{S})$. The *halting problem* for DCMs asks, given a DCM $\mathcal{S}$, whether there is a computation from (s_0, ε) to (s_{halt}, x) in $G(\mathcal{S})$ for some $x \in M^\star$.

Proposition 1 ([6]). *The halting problem for DCMs is undecidable.*

It should be clear that $\mathcal{S}$ has a unique computation. Without loss of generality, we assume that s_{halt} is the only state in S with no outgoing transition. It follows that exactly one of the following must be true: (i) $\mathcal{S}$ has a halting computation; (ii) $\mathcal{S}$ has an infinite computation not reaching s_{halt}; (iii) $\mathcal{S}$ is blocking at some point, i.e., $\mathcal{S}$ is unable to proceed at some state $s \neq s_{halt}$ (with only *read* outgoing transitions) either because the channel is empty or the message at the head of the channel does not match any of the outgoing transitions from s.

Finite-word reactive synthesis for MITL. We now give a reduction from the halting problem for DCMs to $\mathsf{RS}^\star_d(\mathsf{MITL})$. The idea is to devise a suitable MITL formula such that in the corresponding timed game, the environment and the controller are forced to propose actions in turn, according to the semantics of the DCM. Each prefix of the (unique) computation of the DCM is thus encoded as a play, i.e., a finite timed word. More specifically, given a DCM $\mathcal{S}$, we require each play to satisfy the following conditions:

- C1 The action sequence of the play (i.e., omitting all timestamps) is of the form $Nil_C{}^\star\, s_0 a_0 s_1 a_1 \cdots$ where Nil_C is a special action of the controller and $(s_i, a_i, s_{i+1}) \in \Delta$ for each $i \geqslant 0$.
- C2 Each s_i comes with no delay and no two *write* or *read* actions occur at the same time, i.e., if $(a_i, \tau)(s_{i+1}, \tau')(a_{i+1}, \tau'')$ is a substring of the play then $\tau = \tau'$ and $\tau < \tau''$.
- C3 Each $m?$ is preceded exactly 1 time unit (t.u.) earlier by a corresponding $m!$.
- C4 Each $m!$ is followed exactly 1 t.u. later by a corresponding $m?$ if there are actions that occur at least 1 t.u. after the $m!$ in question.

To this end, we construct a formula of the form $\Phi \Rightarrow \Psi$ where Φ and Ψ are conjunctions of the conditions that the environment and the controller must adhere to, respectively. In particular, the environment must propose s_i's according to the transition relation (C1 and C2) whereas the controller is responsible for proposing $\{m!, m? \mid m \in M\}$ properly so that a correct encoding of the writing and reading of messages is maintained (C2, C3, and C4). When both players obey these conditions, the play faithfully encodes a prefix of the computation of $\mathcal{S}$, and the controller wins the play. If the environment attempts to ruin the encoding, the formula will be satisfied, i.e., the play will be winning for the controller. Conversely, if the controller attempts to cheat by, say, reading a message that is not

at the head of the channel, the environment can pinpoint this error (by proposing a special action $\mathit{Check}^{\leftarrow}$) and falsify the formula, i.e., the play will be losing for the controller. In what follows, let $\Sigma_E = S \cup \{\mathit{Check}^{\leftarrow}, \mathit{Check}^{\rightarrow}, \mathit{Lose}, \mathit{Nil}_E\}$, $\Sigma_C = \{m!, m? \mid m \in M\} \cup \{\mathit{Win}, \mathit{Nil}_C\}$, $\varphi_E = \bigvee_{e\in\Sigma_E} e$, $\varphi_C = \bigvee_{c\in\Sigma_C} c$, $\varphi_S = \bigvee_{s\in S} s$, $\varphi_W = \bigvee_{m\in M} m!$, $\varphi_R = \bigvee_{m\in M} m?$ and $\varphi_{WR} = \varphi_W \vee \varphi_R$. Let us now present the formulas $\varphi_1, \varphi_2, \ldots$ and $\psi_1, \psi_2, \ldots$ needed to define Φ and Ψ.

We start by formulas enforcing condition C1. The play should start from s_0, alternate between E-actions and C-actions, and the controller can win the play if the environment does not proceed promptly, and vice versa for the environment:

$$\varphi_1 = \neg\big(\mathit{Nil}_C\,\overline{\mathsf{U}}(\varphi_E \wedge \neg s_0)\big) \qquad \psi_1 = \neg\big(\mathit{Nil}_C\,\overline{\mathsf{U}}(\varphi_C \wedge \neg \mathit{Nil}_C)\big)$$
$$\varphi_2 = \neg\,\overline{\Diamond}(\varphi_E \wedge \bigcirc_{\leqslant 1}\varphi_E) \qquad \psi_2 = \neg\,\overline{\Diamond}(\varphi_C \wedge \bigcirc_{\leqslant 1}\varphi_C)$$
$$\varphi_3 = \neg\,\overline{\Diamond}(\varphi_{WR} \wedge \bigcirc\, \mathit{Win}) \qquad \psi_3 = \neg\,\overline{\Diamond}(\varphi_S \wedge \neg s_{\mathit{halt}} \wedge \bigcirc\, \mathit{Lose})\,.$$

Both players must also comply to the semantics of $\mathcal{S}$:

$$\varphi_4 = \bigwedge_{\substack{(s,a,s')\in\Delta \\ b\notin\{s',\mathit{Check}^{\leftarrow},\mathit{Check}^{\rightarrow}\}}} \neg\,\overline{\Diamond}(s \wedge \bigcirc a \wedge \bigcirc\bigcirc b) \qquad \psi_4 = \bigwedge_{\substack{s\neq s_{\mathit{halt}} \\ \forall s'\ (s,a,s')\notin\Delta}} \neg\,\overline{\Diamond}(s \wedge \bigcirc a)\,.$$

Once the encoding has ended, both players can only propose *Nil* actions:

$$\varphi_5 = \neg\,\overline{\Diamond}\left((s_{\mathit{halt}} \vee \mathit{Check}^{\leftarrow} \vee \mathit{Check}^{\rightarrow} \vee \mathit{Lose} \vee \mathit{Win}) \wedge \Diamond(\varphi_E \wedge \neg \mathit{Nil}_E)\right)$$
$$\psi_5 = \neg\,\overline{\Diamond}\left((s_{\mathit{halt}} \vee \mathit{Check}^{\leftarrow} \vee \mathit{Check}^{\rightarrow} \vee \mathit{Lose} \vee \mathit{Win}) \wedge \Diamond(\varphi_C \wedge \neg \mathit{Nil}_C)\right).$$

For condition C2, we simply state that the environment can only propose delay 0 whereas the controller always proposes a positive delay:

$$\varphi_6 = \neg\,\overline{\Diamond}(\varphi_{WR} \wedge \bigcirc_{>0}\varphi_E) \qquad \psi_6 = \overline{\Box}(\varphi_S \wedge \neg s_{\mathit{halt}} \wedge \bigcirc\,\varphi_{WR} \implies \bigcirc_{>0}\varphi_{WR})\,.$$

Let us finally introduce formulae to enforce conditions C3 and C4. Note that a requirement like 'every write is matched by a read *exactly* one time unit later' is easy to express in MTL, but not so in MITL. Nevertheless, we manage to translate C3 and C4 in MITL by exploiting the game interaction between the players. Intuitively, we allow the cheating player to be punished by the other. Formally, to ensure C3, we allow the environment to play a $\mathit{Check}^{\leftarrow}$ action after any $m?$ to check that this read has indeed occurred 1 t.u. after the corresponding $m!$. Assuming such a $\mathit{Check}^{\leftarrow}$ has occurred, the controller must enforce:

$$\psi^{\leftarrow} = \bigvee_{m\in M} \overline{\Diamond}\left(m! \wedge \overline{\Diamond}_{\leqslant 1}(m? \wedge \bigcirc\, \mathit{Check}^{\leftarrow}) \wedge \overline{\Diamond}_{\geqslant 1}(m? \wedge \bigcirc\, \mathit{Check}^{\leftarrow})\right).$$

Now, to ensure C4, the environment may play a $\mathit{Check}^{\rightarrow}$ action at least 1 t.u. after a write on the channel. If this $\mathit{Check}^{\rightarrow}$ is the first action that occurs more than 1 t.u. after the writing (expressed by the formula $\psi^{\rightarrow}_{\mathit{fst}}$), we must check that the writing has been correctly addressed, i.e., there has been an action exactly 1 t.u. after, *and* this action was the corresponding reading:

$$\psi^{\rightarrow}_{\mathit{fst}} = \overline{\Diamond}(\varphi_W \wedge \overline{\Diamond}_{<1}\,\theta^{\rightarrow}_1 \wedge \overline{\Diamond}_{\geqslant 1}\,\theta^{\rightarrow}_0)$$
$$\psi^{\rightarrow} = \neg\,\overline{\Diamond}(\varphi_W \wedge \overline{\Diamond}_{<1}\,\theta^{\rightarrow}_1 \wedge \overline{\Diamond}_{>1}\,\theta^{\rightarrow}_0) \wedge \psi^{\leftarrow}[\mathit{Check}^{\rightarrow}/\mathit{Check}^{\leftarrow}]$$

where $\psi^{\leftarrow}[\mathit{Check}^{\rightarrow}/\mathit{Check}^{\leftarrow}]$ is the formula obtained by replacing all $\mathit{Check}^{\leftarrow}$ with $\mathit{Check}^{\rightarrow}$ in $\psi^{\leftarrow}$, $\theta_0^{\rightarrow} = \varphi_{WR} \wedge \bigcirc\, \mathit{Check}^{\rightarrow}$ and $\theta_1^{\rightarrow} = \varphi_{WR} \wedge \bigcirc\, \varphi_S \wedge \bigcirc\, \bigcirc\, \theta_0^{\rightarrow}$. In the overall, we consider:

$$\varphi_7 = \bigwedge_{m \in M} \neg\, \overline{\Diamond}(m! \wedge \bigcirc\, \mathit{Check}^{\leftarrow})$$
$$\psi_7 = (\overline{\Diamond}\, \mathit{Check}^{\leftarrow} \Rightarrow \psi^{\leftarrow}) \wedge \left((\overline{\Diamond}\, \mathit{Check}^{\rightarrow} \wedge \psi_{\mathit{fst}}^{\rightarrow}) \Rightarrow \psi^{\rightarrow}\right).$$

Now let $\Phi = \bigwedge_{1 \leqslant i \leqslant 7} \varphi_i$, $\Psi = \bigwedge_{1 \leqslant i \leqslant 7} \psi_i$ and $\Omega = \Phi \Rightarrow \Psi$.

Proposition 2. *Ω is finite-word realisable if and only if either (i) $\mathcal{S}$ has a halting computation, or (ii) $\mathcal{S}$ has an infinite computation not reaching s_{halt}.*[9]

Proof (Sketch). If (i) or (ii) is true, Ω can be realised by the controller faithfully encoding a computation of $\mathcal{S}$. If E proposes $\mathit{Check}^{\leftarrow}$ or $\mathit{Check}^{\rightarrow}$, the play will satisfy ψ_7. Otherwise, if $\mathcal{S}$ has an infinite computation not reaching s_{halt}, the play can grow unboundedly and will satisfy all ψ's, hence Ω.

Conversely, if $\mathcal{S}$ is blocking, then Ω is not realisable. Indeed, either the controller encodes $\mathcal{S}$ correctly, but then at some point it will not be able to propose any action, and will be subsumed by the environment that will play *Lose*. Or the controller will try to cheat, by (1) inserting an action $m?$ not matched by a corresponding $m!$ 1 t.u. earlier, or (2) writing a message $m!$ that will not be read 1 t.u. later. For the first case, the environment can then play $\mathit{Check}^{\leftarrow}$ right after the incorrect $m?$, and the play will violate $\psi^{\leftarrow}$, hence ψ_7 and Ω. For the second case, the environment will play $\mathit{Check}^{\rightarrow}$ after the first action occurring 1 t.u. after the unfaithful $m!$ and the play will violate $\psi^{\rightarrow}$. □

Now let $\Omega' = \Phi \Rightarrow \Psi \wedge \Box(\neg s_{halt})$, i.e., we further require the computation not to reach s_{halt}. The following proposition can be proved almost identically.

Proposition 3. *Ω' is finite-word realisable if and only if $\mathcal{S}$ has an infinite computation not reaching s_{halt}.*

Corollary 1. *$\mathcal{S}$ has a halting computation if and only if Ω is finite-word realisable but Ω' is not finite-word realisable.*

It follows that if $\mathsf{RS}_d^{\star}(\mathsf{MITL})$ is decidable, we can decide whether $\mathcal{S}$ has a halting computation. But the latter is known to be undecidable. Hence:

Theorem 1. $\mathsf{RS}_d^{\star}(\mathsf{MITL})$ *is undecidable.*

Theorem 1 and its proof are the core results from which we will derive all other undecidability results announced at the beginning of the section.

Remark 1. One may show that the RS_d^{ω} problem is undecidable for formulas of the form $\Phi \Rightarrow \Psi$ where Φ and Ψ are conjunctions of $\mathsf{Safety\text{-}MTL}[\mathsf{U}_{0,\infty}]$ formulas by rewriting φ_i's and ψ_i's. This answers an open question of [9].

[9] Observe that the proof does not require any plant (or uses the trivial plant accepting $T\Sigma^{\star}$). This entails undecidability of the 'realisability problem', which is more restrictive than $\mathsf{RS}_d^{\star}$ and another difference with respect to the proof in [4].

BPrecRS and BClockRS for Safety-MTL, coFlat-MTL, and MITL. In the proof of Proposition 2, if $\mathcal{S}$ actually halts, the number of messages present in the channel during the (unique) computation is bounded by a number N. It follows that the strategy of C can be implemented as a bounded-precision controller (with precision $(m, K) = (1, 1)$ and N clocks) or a bounded-clocks controller (with precision $(m, K) = (\frac{1}{N}, 1)$ and a single clock). Corollary 1 therefore holds also for the bounded-precision and bounded-clocks cases, and the $\mathsf{BPrecRS}^{\star}_{d}(\mathsf{MITL})$ and $\mathsf{BClockRS}^{\star}_{d}(\mathsf{MITL})$ problems are undecidable. By further modifying the formulas used in the proof of Proposition 2, we show that the undecidability indeed holds even when we allow only unary non-strict modalities with lower-bound constraints and require the constraints to be exclusively strict or non-strict, hence $\mathsf{BPrecRS}^{\star}_{d}$ and $\mathsf{BClockRS}^{\star}_{d}$ are undecidable too on $\mathsf{Open\text{-}MITL}^{\mathrm{ns}}[\Diamond_{\infty}]$ and $\mathsf{Closed\text{-}MITL}^{\mathrm{ns}}[\Diamond_{\infty}]$. This entails undecidability in the *undesired specifications* case because the negation of an $\mathsf{Open\text{-}MITL}^{\mathrm{ns}}[\Diamond_{\infty}]$ is a $\mathsf{Closed\text{-}MITL}^{\mathrm{ns}}[\Diamond_{\infty}]$ formula and vice-versa. Finally, we can extend our proofs to the infinite words case, hence:

Theorem 2. $\mathsf{RS}^{b}_{s}(\mathsf{L})$, $\mathsf{IRS}^{b}_{s}(\mathsf{L})$, $\mathsf{BPrecRS}^{b}_{s}(\mathsf{L})$ *and* $\mathsf{BClockRS}^{b}_{s}(\mathsf{L})$ *are undecidable for* $\mathsf{L} \in \{\mathsf{Open{-}MITL}^{\mathrm{ns}}[\Diamond_{\infty}], \mathsf{Closed{-}MITL}^{\mathrm{ns}}[\Diamond_{\infty}]\}$, $s \in \{u, d\}$ *and* $b \in \{\star, \omega\}$.

This result extends the previous undecidability proofs of [12] ($\mathsf{RS}^{\omega}_{d}(\mathsf{ECL})$ is undecidable), and of [13] ($\mathsf{IRS}^{\star}_{d}(\mathrm{TA})$ and $\mathsf{IRS}^{\star}_{u}(\mathrm{TA})$ are undecidable). In light of these previous works, our result is somewhat surprising as the undecidability proof in [13] is via a reduction from the universality problem for timed automata, yet this universality problem becomes decidable when all constraints are strict [17].

Finally, it remains to handle the cases of Safety-MTL and coFlat-MTL. Contrary to the case of MTL, the infinite-word satisfiability problem is decidable for Safety-MTL [19] and the infinite-word model-checking problem is decidable for both Safety-MTL [19] and coFlat-MTL [5]. Nevertheless, our synthesis problems remain undecidable for these fragments. In particular, the result on Safety-MTL answers an open question of [4] negatively:

Theorem 3. $\mathsf{RS}^{b}_{s}(\mathsf{L})$, $\mathsf{IRS}^{b}_{s}(\mathsf{L})$, $\mathsf{BPrecRS}^{b}_{s}(\mathsf{L})$ *and* $\mathsf{BClockRS}^{b}_{s}(\mathsf{L})$ *are undecidable for* $\mathsf{L} \in \{\mathsf{Safety\text{-}MTL}, \mathsf{coFlat\text{-}MTL}\}$, $s \in \{u, d\}$ *and* $b \in \{\star, \omega\}$.

4 Bounded-resources synthesis for MITL properties

We have now characterised rather precisely the decidability border for MITL synthesis problems. In light of these results, we focus now on $\mathsf{BResRS}^{\star}_{d}(\mathsf{MITL})$ (since MITL is closed under complement, one can derive an algorithm for $\mathsf{BResRS}^{\star}_{u}(\mathsf{MITL})$ from our solution). Recall that the algorithm of D'Souza and Madhusudan [13], associated with the translation of MITL into TA [2] yields a 3EXPTIME procedure for these two problems. Unfortunately this procedure is unlikely to be amenable to efficient implementation. This is due to the translation from MITL to TA and the need to determinise a region automaton, which is known to be hard in practice. On the other hand, Bouyer *et al.* [4] present a procedure for

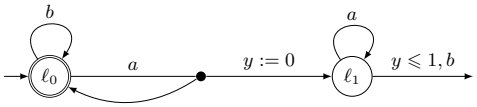

Fig. 3. An OCATA (with single clock y) accepting the language of $\square(a \Rightarrow \Diamond_{\leqslant 1} b)$.

$\mathsf{BResRS}^{\star}_{d}(\mathsf{MTL})$ (which can thus be applied to MITL requirements). This algorithm is on-the-fly, in the sense that it avoids, if possible to build a full automaton for the requirement; and thus more likely to perform well in practice. Unfortunately, being designed for MTL, its running time can only be bounded above by a non-primitive recursive function. We present now an algorithm for $\mathsf{BResRS}^{\star}_{d}(\mathsf{MITL})$ that combines the advantages of these two previous solutions: it is *on-the-fly* and runs in 3EXPTIME. To obtain an on-the-fly algorithm, Bouyer *et al.* use *one-clock alternating automata* (OCATA) instead of TA to represent the MITL requirement. We follow the same path, but rely on the newly introduced *interval-based semantics* [7] for these automata, in order to mitigate the complexity. Let us now briefly recall these two basic ingredients.

OCATA and interval semantics. Alternating timed automata [19] extend (non-deterministic) timed automata by adding *conjunctive transitions*. Intuitively, conjunctive transitions spawn several copies of the automaton that run in parallel from the target states of the transition. A word is accepted iff *all* copies accept it. An example is shown in Fig. 3, where the conjunctive transition is the hyperedge starting from ℓ_0. In the classical semantics, an execution of an OCATA is a sequence of set of states, named *configurations*, describing the current location and clock valuation of all active copies. For example, a prefix of execution of the automaton in Fig. 3 would start in $\{(\ell_0, 0)\}$ (initially, there is only one copy in ℓ_0 with the clock equal to 0); then $\{(\ell_0, 0.42)\}$ (after letting 0.42 time units elapse); then $\{(\ell_0, 0.42), (\ell_1, 0)\}$ (after firing the conjunctive transition from ℓ_0), etc. It is well-known that all formulas φ of MTL (hence, also MITL) can be translated into an OCATA A_φ that accepts the same language [19] (with the classical semantics); and with a number of locations linear in the number of subformulas of φ. This translation is thus straightforward. This is the key advantage of OCATA over TA: the complexity of the MITL formula is shifted from the syntax to the semantics—what we need for an on-the-fly algorithm.

Then; in the *interval semantics* [7], valuations of the clocks are not *points* anymore but *intervals*. Intuitively, intervals are meant to approximate sets of (punctual) valuations: $(\ell, [a, b])$ means that there *are* clock copies with valuations a and b in ℓ, and that there *could be* more copies in ℓ with valuations in $[a, b]$. In this semantics, we can also *merge* two copies $(\ell, [a_1, b_1])$ and $(\ell, [a_2, b_2])$ into a single copy $(\ell, [a_1, b_2])$ (assuming $a_1 \leqslant b_2$), in order to keep the number of clock copies below a fixed threshold K. It has been shown [7] that, when the OCATA has been built from an MITL formula, the interval semantics is sufficient to retain the language of the formula, with a number of copies which is at most doubly exponential in the size of the formula.

Table 1. Experimental results on the scheduling problem: realisable instances on the left, non-realisable on the right.

T	n	# clocks	exec. time (sec) / #nodes
1	1	0	46 / 52
1	1	1	199 / 147
1	1	2	4,599 / 1,343
2	2	1	2,632 / 645
2	2	2	18,453 / 2,358
3	3	1	182,524 / 2,297
3	3	2	>5min
4	4	0	54,893 / 667
4	4	1	>5min

T	n	# clocks	exec. time (sec) / #nodes
2	1	0	77 / 84
2	1	1	824 / 311
2	1	2	3,079 / 1,116
3	2	1	17,134 / 1698
3	2	2	>5min
4	3	0	10,621 / 540
4	3	1	>5min

Sketch of the algorithm. Equipped with these elements, we can now sketch our algorithm for $\mathsf{BResRS}^{\star}_{d}(\mathsf{MITL})$. Starting from an MITL formula φ, a plant $\mathcal{P}$ and a granularity $\mu = (X, m, K)$, we first build, in polynomial time, an OCATA $A_{\neg\varphi}$ accepting $\mathcal{L}(\neg\varphi)$. Then, we essentially adapt the technique of Bouyer *et al.* [4], relying on the interval semantics of OCATA instead of the classical one. This boils down to building a tree that unfolds the parallel execution of $A_{\neg\varphi}$ (in the interval semantics), $\mathcal{P}$ and all possible actions of a μ-granular controller (hence the *on-the-fly* algorithm). Since the granularity is fixed, there are only finitely many possible actions (i.e., guards and resets on the controller clocks) for the controller at each step. We rely on the region construction to group the infinitely many possible valuations of the clocks into finitely many equivalence classes that are represented using 'region words' [19]. The result is a finitely branching tree that might still have infinite branches. We stop developing a branch once a global configuration (of $A_{\neg\varphi}$, $\mathcal{P}$, and the controller) repeats on the branch. By the region construction *and* the interval semantics, this will happen on all branches, and we obtain a *finite tree* of size at most triply exponential. This tree can be analysed (using backward induction) as a game with a safety objective for the controller: to avoid the nodes where $\mathcal{P}$ and $A_{\neg\varphi}$ accept at the same time. The winning strategy yields, if it exists, a correct controller.

Experimental results. We have implemented our procedure in Java, and tested it over a benchmark related to a scheduling problem, inspired by an example of [9]. This problem considers n machines, and a list of jobs that must be assigned to the machines. A job takes T time units to finish. The plant ensures that at least one time unit elapses between two job arrivals (which are uncontrollable actions). The specification asks that the assignment be performed in 1 time unit, and that each job has T time units of computation time. We tested this example with $T = n$, in which case the specification is realisable (no matter the number of clocks, which we make vary for testing the prototype efficiency), and with $T = n + 1$, in which case it is not. Table 1 summarises some of our results.

These results show that our prototypes can handle small but non-trivial examples. Unfortunately—as expected by the high complexities of the algorithm—they do not scale well. As future works, we will rely on the well-quasi orderings defined in [4] to introduce heuristics in the spirit of the antichain techniques [15]. Second, we will investigate zone-based versions of this algorithm to avoid the state explosion which is inherent to region based techniques.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *T.C.S*, 126(2):183–235, 1994.
2. R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.
3. A. Bohy, V. Bruyère, E. Filiot, N. Jin, and J. Raskin. Acacia+, a tool for LTL synthesis. In *CAV'12*, LNCS 7358, Springer.
4. P. Bouyer, L. Bozzelli, and F. Chevalier. Controller synthesis for MTL specifications. In *CONCUR'06*, LNCS 4137, Springer.
5. P. Bouyer, N. Markey, J. Ouaknine, and J. Worrell. The cost of punctuality. In *LICS'07*, pages 109–120. IEEE.
6. D. Brand and P. Zafiropulo. On communicating finite state machines. *J. ACM*, 30:323–342, 1983.
7. T. Brihaye, M. Estiévenart, and G. Geeraerts. On MITL and alternating timed automata. In *FORMATS'13*, LNCS 8053, Springer.
8. T. Brihaye, M. Estiévenart, G. Geeraerts, H.-M. Ho, B. Monmege, and N. Sznajder. Real-time Synthesis is Hard! (full version) arXiv:1606.07124, 2016.
9. P. E. Bulychev, A. David, K. G. Larsen, and G. Li. Efficient controller synthesis for a fragment of $\mathsf{MTL}_{0,\infty}$. *Acta Inf.*, 51(3-4):165–192, 2014.
10. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR'05*, LNCS 3653, Springer.
11. L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *CONCUR'03*, LNCS 2761, Springer.
12. L. Doyen, G. Geeraerts, J.-F. Raskin, and J. Reichert. Realizability of real-time logics. In *FORMATS'09*, LNCS 5813, Springer.
13. D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *STACS'02*, LNCS 2285, Springer.
14. M. Estiévenart. Verification and synthesis of MITL through alternating timed automata. PhD. thesis, Université de Mons, 2015.
15. E. Filiot, N. Jin, and J. Raskin. An antichain algorithm for LTL realizability. In *CAV'09*, LNCS 5643, Springer.
16. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
17. J. Ouaknine and J. Worrell. Universality and language inclusion for open and closed timed automata. In *HSCC'03*, LNCS 2623, Springer.
18. J. Ouaknine and J. Worrell. Safety metric temporal logic is fully decidable. In *TACAS'06*, LNCS 3920, Springer.
19. J. Ouaknine and J. Worrell. On the decidability and complexity of metric temporal logic over finite words. *LMCS*, 3(1), 2007.
20. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *ICALP'89*, LNCS 372, Springer.
21. J.-F. Raskin. *Logics, automata and classical theories for deciding real time.* PhD thesis, FUNDP (Belgium), 1999.