One-Clock Synthesis Problems

Sławomir Lasota
University of Warsaw, Poland

Mathieu Lehaut
University of Warsaw, Poland

Julie Parreaux
Univ Rennes IRISA, Inria, CNRS, France

Radosław Piórkowski
University of Oxford, UK

Abstract

We study a generalisation of Büchi-Landweber games to the timed setting. The winning condition is specified by a non-deterministic timed automaton, and one of the players can elapse time. We perform a systematic study of synthesis problems in all variants of timed games, depending on which player's winning condition is specified, and which player's strategy (or controller, a finite-memory strategy) is sought.

As our main result we prove ubiquitous undecidability in all the variants, both for strategy and controller synthesis, already for winning conditions specified by one-clock automata. This strengthens and generalises previously known undecidability results. We also fully characterise those cases where finite memory is sufficient to win, namely existence of a strategy implies existence of a controller.

All our results are stated in the *timed* setting, while analogous results hold in the *data* setting where one-clock automata are replaced by one-register ones.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Automata over infinite objects; Theory of computation → Quantitative automata; Theory of computation → Logic and verification

Keywords and phrases timed automata, register automata, Büchi-Landweber games, Church synthesis problem, reactive synthesis problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.64

Related Version *Extended Version*: `https://arxiv.org/abs/2601.04902` [26]

Funding *Sławomir Lasota*: Partially supported by ERC grant INFSYS, agreement no. 950398 and by the NCN grant 2024/55/B/ST6/01674.
Mathieu Lehaut: Partially supported by the NCN grant 2021/41/B/ST6/00535.
Julie Parreaux: Partially supported by the ANR project BisoUS (ANR-22-CE48-0012).

1 Introduction

Nondeterministic timed automata (NTA) are one of the most widespread models of real-time reactive systems with a huge literature. They consist of finite automata extended with real-valued clocks which can be reset and compared by inequality constraints. Since the seminal paper showing PSpace-completeness of the reachability problem [2], NTA found their way to the automatic verification of timed systems, eventually leading to mature tools such as UPPAAL [5], UPPAAL Tiga (timed games) [11], and PRISM (probabilistic timed automata) [24]. One of the restrictions of this model is the lack of determinisation and closure under complementation of NTA languages, as well as undecidability of universality/inclusion problems [2]. One recovers decidability for a restricted subclass of NTA with one clock (NTA_1) [31], and even for alternating timed automata with one clock [25].

43rd International Symposium on Theoretical Aspects of Computer Science (STACS 2026).
Editors: Meena Mahajan, Florin Manea, Annabelle McIver, and Nguyễn Kim Thắng
Article No. 64; pp. 64:1–64:21


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Deterministic timed automata (DTA) form a strict subclass of NTA where the next configuration is uniquely determined by the current one and the timed input symbol. This class enjoys stronger properties, such as decidable inclusion problems and complementability [2], and it is used in several applications, such as test generation [30], fault diagnosis [8], learning [37, 36]; timed games [3, 22, 9], and recognisability of timed languages [27].

Timed games and synthesis problems. There are many variants of timed games in the literature, depending on whether the players must enforce a non-Zeno play, who controls the elapse of time, concurrent actions, winning conditions, etc. [3, 22, 9, 38, 28, 4, 16, 14, 33, 21]. Following prior works [13, 32], we consider asymmetric (only one player can elapse time), infinite-duration turn-based games that can be considered as timed generalisation of Büchi-Landweber games [10]. There are two players, called Timer and Monitor, who play taking turns in a strictly alternating fashion. In the ith round, Timer chooses a letter a_i from a finite alphabet along with a nonnegative rational time delay τ_i, and Monitor responds with a letter b_i from a finite alphabet. Together, the players generate an infinite play $\pi = (a_1, b_1, \tau_1)\,(a_2, b_2, \tau_2)\cdots$. The winning set of either Timer or Monitor is specified by a *nondeterministic* timed automaton. For comparison, the easier special case where the winning set is given by a *deterministic* or *history deterministic* timed automaton has been previously studied (e.g., [16, 7]). Given our undecidability results, we do not consider a more general symmetric variant in which both players may elapse time.

We investigate the *timed reactive synthesis* problem, which asks if a given player has a strategy ensuring that every play that conforms to the strategy is winning for that player. We distinguish four variants of this problem, depending on which player's winning set is specified (note the lack of complement closure of NTA languages) and for which player the winning strategy is sought. We also study the *timed Church synthesis* problem, which asks if a given player has a winning finite-memory strategy (controller). For Monitor, a controller is a DTA whose transitions output letters from Monitor's alphabet; for Timer, it is a DFA whose transitions output letters from Timer's alphabet along with time delays.

This study of timed generalisations of Büchi–Landweber games was initiated in [13] in the setting where Timer's winning set is specified by an NTA, and the goal is to synthesise a controller for Monitor. The main result of that work is the decidability of the *resource-bounded* timed Church synthesis problem, in which one seeks a controller using at most k clocks, for a fixed k. Subsequently, [32] established the undecidability of the unrestricted (resource-unbounded) version, even when Timer's winning set is given by a two-clock NTA and Monitor's controller is sought. The decidability of this problem for NTA_1 (one-clock NTA) winning sets remained open, as did the status of other variants and of timed reactive synthesis. The present paper resolves all of these open questions in the negative.

Contribution. Given as many as eight different decision problems, one could expect a complex decidability/complexity landscape. As our main technical contribution, we show that this is not the case: all eight decision problems are undecidable already in the simplest case where the winning sets are specified by NTA_1. These undecidability results significantly strengthen and generalise previously known lower bounds. They also demonstrate that restricting to NTA_1 does not lead to recovery of decidability of synthesis (game solving) problems, similarly as restricting to NTA_1 does not yield decidability of language universality and related problems over infinite words (as opposed to universality and related problems for NTA_1 languages of finite timed words [31]).

Proving undecidability for eight problems required four reductions. The cases where Monitor's winning set is specified by an NTA_1 are easily shown undecidable by reductions from the NTA_1 universality and sampled universality problems [1, 25]. Undecidability proofs in the other case, where Timer's winning set is specified, constitute the technical core of the paper, and proceed by reductions from two undecidable problems for lossy counter machines: boundedness and repeated reachability [29, 34].

Finally, we also address the question of when finite memory is sufficient to win, that is when existence of a strategy implies existence of a controller. On one hand, we prove this finite-memory property for the player whose winning set is specified: for all NTA_1 specifications if this player is Monitor, and only for the restricted case of *reachability* NTA_1 specifications if this player is Timer. On the other hand, we demonstrate that the finite-memory property fails in all other cases: it is not satisfied in general by Timer when his winning set is specified; and the property fails for both Timer and Monitor when the opponent's winning set is specified, even in case of reachability NTA_1 specifications.

It is well known that NTA exhibit close similarity to *nondeterministic register automata* (NRA), known also as finite-memory automata [23] (see e.g. [20] for a formal connection between the models). Register automata input data values (in place of timestamps) and use registers (in place of clocks) to store data values ([35] is an excellent survey of automata models for data setting). For many language-related problems, like emptiness (reachability), universality or inclusion, register automata admit exactly the same (un)decidability results [15] that are known for timed automata [31, 25]. Language closure properties are also analogous in both settings. Confirming these deep similarities further, [32] proves undecidability of the *timed Church synthesis* problem for winning sets specified by NTA with two clocks, as well as undecidability of the *data Church synthesis* for winning sets specified by NRA with two registers. Similar studies of data generalisation of Büchi-Landweber games were also conducted independently in [17] and works cited therein [19, 18]. All our undecidability results transfer from the timed to the data setting: undecidability holds for all the eight variants of *data reactive/Church synthesis* problems, already when winning sets are specified by NRA_1 (NRA with one register). (These further results exceed the scope of the present paper and are planned to be included in the forthcoming full version of the paper.)

Outline. We start by defining the setting of timed games and the synthesis problems (Section 2). We also discuss the finite-memory property there. Section 3 is a warm-up where we deal with the easy cases of synthesis problems (Monitor's winning set is specified). Then in Section 4 we define the undecidable problems for lossy counter machines, to be used in the main technical part of the paper, Sections 5 and 6. In the two latter sections we provide the undecidability proofs in the hard case where Timer's winning set is specified. The last section contains final remarks. All omitted proofs can be found in the long version of this paper [26].

2 Timed synthesis problems

Let $\mathbb{Q}_{\geq 0}$ and $\mathbb{N}_{>0}$ be nonnegative rational numbers, and positive integers, respectively. We let $\mathcal{X}$ be a finite set of variables called *clocks*. A *valuation* is a mapping $\nu\colon \mathcal{X} \to \mathbb{Q}_{\geq 0}$ (we prefer to use rational values instead of real ones). For a valuation ν, a delay $\tau \in \mathbb{Q}_{\geq 0}$ and a subset $Y \subseteq \mathcal{X}$ of clocks, we define the valuation $\nu + \tau$ as $(\nu + \tau)(x) = \nu(x) + \tau$, for all $x \in \mathcal{X}$, and the valuation $\nu[Y := 0]$ as $(\nu[Y := 0])(x) = 0$ if $x \in Y$, and $(\nu[Y := 0])(x) = \nu(x)$ otherwise. A *guard* on clocks $\mathcal{X}$ is a conjunction of atomic constraints of the form $x \bowtie c$,

where $x \in \mathcal{X}$, $\bowtie \in \{\leq, <, =, >, \geq\}$ and $c \in \mathbb{N}$. An empty guard is denoted by $\top$. A valuation ν *satisfies* an atomic constraint $x \bowtie c$ if $\nu(x) \bowtie c$. The satisfaction relation is extended to all guards g naturally, and denoted by $\nu \models g$. We let $\mathrm{Guard}(\mathcal{X})$ denote the set of guards over $\mathcal{X}$.

Let Σ be a finite alphabet. By a *timed word* over Σ we mean a finite or infinite sequence $w = (a_1, t_1)(a_2, t_2)\cdots \in (\Sigma \times \mathbb{Q}_{\geq 0})^\omega$ where the *timestamps* t_i are monotone: $t_1 \leq t_2 \leq t_3 \leq \cdots$. Pairs $(a, t) \in \Sigma \times \mathbb{Q}_{\geq 0}$ are sometimes called *timed letters*. Unless stated otherwise, we work with infinite timed words. The sequence of timestamps is determined uniquely by the sequence of *delays* $\tau_1, \tau_2, \ldots \in \mathbb{Q}_{\geq 0}$, where $\tau_i = t_i - t_{i-1}$ (assuming $t_0 = 0$). The *untiming* of w, denoted by $\mathrm{untime}(w)$, is the word $a_1 a_2 \ldots \in \Sigma^\omega$ obtained by removing timestamps. A timed language is any set of timed words over a fixed alphabet Σ.

Timed automata. A *non-deterministic timed automaton* (NTA) $\mathcal{A} = \langle L, \mathcal{X}, \Sigma, L_i, L_f, \Delta\rangle$ consists of L, a finite set of *locations* with $L_i, L_f \subseteq L$ denoting the sets of initial and accepting locations, respectively; $\mathcal{X}$, a finite set of clocks; Σ, a finite alphabet; and $\Delta \subseteq L \times \Sigma \times \mathrm{Guard}(\mathcal{X}) \times 2^{\mathcal{X}} \times L$, a finite set of transitions. A transition $(\ell, a, g, Y, \ell') \in \Delta$ is written as $\ell \xrightarrow{a,g,Y} \ell'$, omitting g and Y whenever they are $\top$ or $\emptyset$, respectively. A set A in place of a specifies a set of transitions. The *semantics* of $\mathcal{A} \in \mathrm{NTA}$ is a timed transition system $[\![\mathcal{A}]\!] = (Q, Q_I, \to)$ such that: $Q = L \times (\mathbb{Q}_{\geq 0})^{\mathcal{X}}$ is the set of *configurations* (i.e., location-valuation pairs), $Q_I = L_i \times \{0^{\mathcal{X}}\}$ is the set of initial configurations, and $\to$ is the set of *edges*. We have $(\ell, \nu) \xrightarrow{\delta,\tau} (\ell', \nu')$ if and only if $\delta = (\ell, a, g, Y, \ell') \in \Delta$ is a transition of $\mathcal{A}$ such that $\nu + \tau \models g$, and $\nu' = (\nu + \tau)[Y := 0]$. A *run* ρ in $\mathcal{A}$ is a sequence of edges $\rho = (\ell_1, \nu_1) \xrightarrow{\delta_1,\tau_1} (\ell_2, \nu_2) \xrightarrow{\delta_2,\tau_2} \ldots$ of $[\![\mathcal{A}]\!]$ such that $\ell_1 \in L_i$. A timed word $w = (a_1, t_1)(a_1, t_2)\ldots$ over Σ *labels* a run ρ of $\mathcal{A}$ when, for all $i \in \mathbb{N}_{>0}$, $\delta_i = (\ell_i, a_i, g, Y, \ell_{i+1})$ and $t_i = \sum_{j=1}^{i} \tau_j$. We adopt *Büchi* acceptance: a run is *accepting* if is visits an accepting location infinitely often, in which case we say that the word that labels that run is *accepted* by $\mathcal{A}$. The *language* of $\mathcal{A} \in \mathrm{NTA}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of timed words accepted by $\mathcal{A}$.

We also use *reachability* acceptance, where a run is accepting once it visits an accepting location at least once (like in the setting of finite words). With this acceptance, the language L of *infinite* timed words accepted by an NTA is determined uniquely by the language L' of *finite* timed words having a run ending in an accepting location. We write $L = \mathrm{Reach}(L')$. So defined *reachability NTA* can be seen as a (strict) subclass of NTA, denoted as *reach-NTA*.

Apart from reach-NTA, we consider also other subclasses of NTA. For instance NTA_k consists of nondeterministic timed automata with a fixed number $k = |\mathcal{X}|$ of clocks. DTA is the class of *deterministic* timed automata, where $|L_i| = 1$ and for all locations $\ell \in L$ and letters $a \in \Sigma$, the guards g appearing in transitions of the form $(\ell, a, g, \_, \_)$ form a partition of $\mathbb{Q}_{\geq 0}^{|\mathcal{X}|}$. We also consider a superclass $\mathrm{NTA}^{\mathrm{res}}$ of *1-resetting* timed automata (defined in Section 6), an extension by a limited form of ε-transitions that reset a clock every time it equals 1. The class of NTA with ε-transition is strictly more expressive than NTA [6]; we discuss this choice at the end of this section, and in Sections 6 and 7.

We combine the notation for sub- and superclasses and write reach-NTA_1, reach-$\mathrm{NTA}_1^{\mathrm{res}}$, for the one-clock automata from reach-NTA, and for the 1-resetting extension thereof.

Given L a timed language, we denote by $\widehat{L}$ its *complement*. The *universality problem* asks, given $\mathcal{A}$, if $\widehat{\mathcal{L}(\mathcal{A})} = \emptyset$. This problem is known to be undecidable for NTA_1 [25].

Timed games. In this paper we consider turn-based games between two players called here Timer and Monitor, where a winning condition is given by an NTA_1 (resp. $\mathrm{NTA}_1^{\mathrm{res}}$).

▶ **Definition 1.** *A* timed game $\mathcal{G} = \langle \mathrm{T}, \mathrm{M}, \mathcal{A}, \mathrm{Owner}, \mathrm{Agent} \rangle$ *consists of finite alphabets* T, M *of* Timer *and* Monitor*, respectively,* $\mathcal{A} \in NTA_1$ *(resp.* $\mathcal{A} \in NTA_1^{res}$*) a timed automaton over the product alphabet* $\mathrm{T} \times \mathrm{M}$ *whose language* $\mathcal{L}(\mathcal{A})$ *defines the winning condition, and* $\mathrm{Owner}, \mathrm{Agent} \in \{\mathrm{Timer}, \mathrm{Monitor}\}$.

Intuitively, Owner specifies the player who wins a play if it belongs to $\mathcal{L}(\mathcal{A})$. As NTA are not stable by complement [2], the choice of the owner of the winning condition is not innocuous, and the winning condition of the opponent may no longer be an NTA language. We investigate decision problems asking about existence of a winning strategy of Agent. Again, the choice of Agent is not innocuous, as we do not know if the games studied by us are determined. In view of our undecidability results, we do not consider a symmetric variant of timed games where both players would submit time delays, as this variant would generalise the above one.

A timed game proceeds in rounds. Each ith round ($i \in \mathbb{N}_{>0}$) starts by Timer's choice of $a_i \in \mathrm{T}$ and a delay $\tau_i \in \mathbb{Q}_{\geq 0}$, followed by a response $b_i \in \mathrm{M}$ of Monitor. This results in a *play* which is a timed word $(a_1, b_1, t_1)\,(a_2, b_2, t_2) \cdots$ over the alphabet $\mathrm{T} \times \mathrm{M}$, where $t_i = \sum_{j=1}^{i} \tau_j$. Owner wins the play if it belongs to $\mathcal{L}(\mathcal{A})$, otherwise its opponent wins.

A *strategy* for a player is a function that gives its next move as a function of the moves of its opponent up to now. Formally, a strategy for Timer is a function $\sigma_{\mathrm{T}} \colon \mathrm{M}^* \to \mathrm{T} \times \mathbb{Q}_{\geq 0}$ whereas a strategy for Monitor is a function $\sigma_{\mathrm{M}} \colon (\mathrm{T} \times \mathbb{Q}_{\geq 0})^+ \to \mathrm{M}$. We say that a play $w = (a_1, b_1, t_1)\,(a_2, b_2, t_2)\,\ldots$ *conforms* to Timer's strategy σ_{T} (resp. Monitor's strategy σ_{M}) if all (timed) letters in it conform to the strategy's output: $(a_i, \tau_i) = \sigma_{\mathrm{T}}(b_1 \cdots b_{i-1})$ for all $i \in \mathbb{N}_{>0}$ (resp., $b_i = \sigma_{\mathrm{M}}((a_1, t_1) \cdots (a_i, t_i))$ for all $i \in \mathbb{N}_{>0}$). A strategy is *winning* for a player if and only if every play w that conforms to it is winning for that player (i.e., $w \in \mathcal{L}(\mathcal{A})$ if and only if this player is the owner).

A *controller* is a finite-memory strategy represented by a DTA with outputs. Specifically, a Monitor's controller is a DTA with outputs from M, whose transitions are of the form $\delta = (\ell, a, g, Y, \ell', b)$, where $b \in \mathrm{M}$ is the output. The controller induces the strategy σ_{M} that maps $w = (a_1, t_1) \cdots (a_i, t_i)$ to the last output of the run over w. For Timer, the notion of controller is more tricky since it needs to produce timestamps. We let Timer's controllers output pairs $(a, \tau) \in \mathrm{T} \times \mathbb{Q}_{\geq 0}$, where τ is interpreted as the next *delay*. Formally, it is defined as a DFA (DTA with no clocks) with outputs from $\mathrm{T} \times \mathbb{Q}_{\geq 0}$, whose transitions are of the form $\delta = (\ell, b, \ell', a, \tau)$, where $(a, \tau) \in \mathrm{T} \times \mathbb{Q}_{\geq 0}$ is the output, additionally equipped with an *initial move* $(a_1, \tau_1) \in \mathrm{T} \times \mathbb{Q}_{\geq 0}$. Note the apparent restriction of Timer's controllers, compared to general strategies, namely the set of delays used by a controller is finite. A controller induces the strategy σ_{T} that maps the empty word to (a_1, τ_1), and a nonempty word $b_1 \cdots b_i$ to the last output of the run over w.

Timed synthesis problems. We investigate decision problems to determine whether Agent wins, i.e., whether Agent has a winning strategy. We distinguish four cases, depending on the choice of Owner $\in \{\mathrm{Timer}, \mathrm{Monitor}\}$ and Agent $\in \{\mathrm{Timer}, \mathrm{Monitor}\}$. Furthermore, in each of the four cases we consider two distinct decision problems: given a timed game,

- the *timed reactive synthesis* asks if Agent has a winning strategy;
- the *timed Church synthesis* asks if Agent has a winning controller.

The problems coincide in some cases (see Table 1 for a summary). Most importantly, if Monitor is both the owner of the winning condition, and also the agent of the synthesis problem, then whenever it has a winning strategy it also has a controller. Likewise for Timer, but only in the restricted case of reach-NTA_1 winning conditions:

Table 1 Reactive vs. Church synthesis.

	Agent = Timer	Agent = Monitor
Owner = Timer	coincide for $\mathcal{A} \in$ reach-NTA$_1$, but not for all $\mathcal{A} \in$ NTA$_1$	do not coincide, even for $\mathcal{A} \in$ reach-NTA$_1$
Owner = Monitor	do not coincide, even for $\mathcal{A} \in$ reach-NTA$_1$	coincide for all $\mathcal{A} \in$ NTA$_1$

▶ **Theorem 2.** *The two synthesis problems coincide, meaning that if* Agent *has a winning strategy then it has a winning controller, in the following cases:*

1. Owner = Agent = Timer *and* $\mathcal{A} \in$ *reach-NTA*$_1$.
2. Owner = Agent = Monitor *and* $\mathcal{A} \in$ *NTA*$_1$.

In all the remaining cases the two synthesis problems do not coincide, namely existence of a winning strategy does not imply existence of a controller: first, when Timer is both the owner of the winning condition, and also the agent of the synthesis problem, but the winning condition is an arbitrary NTA$_1$ language; second, when the owner is different from the agent, even in the case of reachability NTA$_1$ winning conditions:

▶ **Theorem 3.** *The two synthesis problems may not coincide in the following cases:*

1. Owner $\neq$ Agent *and* $\mathcal{A} \in$ *reach-NTA*$_1$.
2. Owner = Agent = Timer *and* $\mathcal{A} \in$ *NTA*$_1$.

Proof.

1. We start with the case Owner = Timer and Agent = Monitor. Let Timer's alphabet be trivial (singleton), and hence omitted below; Timer thus essentially chooses only timestamps. Let Monitor's alphabet $\mathrm{M} = \{a, b\}$, and let Timer's winning condition contain timed words containing some letter (a or b) at distance 1, that is timed words that contain both (a, t) and $(a, t+1)$, or both (b, t) and $(b, t+1)$, for some $t \in \mathbb{Q}_{\geq 0}$. The condition is readily seen to be recognised by a reachability NTA$_1$. Monitor has a winning strategy in this game: it wins by always playing the same letter (say a) *unless* timestamp $t-1$ appeared earlier in the play, in which case it plays the opposite of the letter that was played there. On the other hand, Monitor has no winning controller, as winning requires storing unboundedly many different timestamps (together with letters used at them).
   For the case Owner = Monitor and Agent = Timer, we choose both alphabets to be trivial (and hence omitted below). The play is thus just a monotonic sequence $t_1 \leq t_2 \leq \cdots$ of timestamps, chosen by Timer. Let Monitor's winning condition consist of those sequences which either exceed 1 at some point ($t_i > 1$ for some $i \in \mathbb{N}_{>0}$), or repeat a timestamp ($t_i = t_{i+1}$ for some $i \in \mathbb{N}_{>0}$). Both conditions, and hence their union, are recognised by reachability NTA$_1$. Furthermore, Timer has a winning strategy by producing a Zeno word bounded by 1, but Timer has no winning controller. Indeed, in order to win, Timer should use strictly positive delays only, and therefore, since every controller uses only finitely many different delays, the bound 1 is inevitably exceeded.
2. Consider the case Owner = Agent = Timer. We use the same idea as in the previous case: Timer can win only by producing a Zeno word. Let Timer's alphabet be trivial (hence omitted below); Timer thus essentially chooses only timestamps. Let Monitor's alphabet be $\mathrm{M} = \{\checkmark, \lightning\}$, and let Timer's winning condition be the union of two sets: the timed words $w = (b_1, t_1)(b_2, t_2) \cdots \in (\mathrm{M} \times \mathbb{Q}_{\geq 0})^\omega$ that never exceed 1 labelled exclusively by $\checkmark$, i.e., $t_i < 1$ and $b_i = \checkmark$ for all $i \in \mathbb{N}_{>0}$; and the timed words such that for some $i \in \mathbb{N}_{>0}$ we have $t_i < t_{i+1}$, $b_1 = \cdots = b_i = \checkmark$ and $b_{i+1} = \lightning$. Timer has a winning strategy by

producing a *strictly monotonic* Zeno word bounded by 1. On the other hand a violation of strict monotonicity, namely the equality $t_i = t_{i+1}$, is punished by Monitor playing $b_{i+1} = \chi$. Therefore, Timer has no winning controller for the same reason as in the previous case. ◀

Summary of results. As our main contribution, we obtain undecidability in all cases of both the reactive and Church synthesis problems, even under the very restricted case where the timed winning condition is specified by an NTA_1 (or by a reach-$\mathrm{NTA}_1^{\mathrm{res}}$ in one of the variants, see Table 2 for a summary). Our reductions in the case of the reactive synthesis problem work uniformly, regardless of which player is the Agent. Therefore Table 2 has six rather than eight entries. These six cases require four different reductions, and hence each table entry indicates the problem we reduce from. When Timer is the owner of the winning condition, some undecidable problems for lossy counter machines (LCM) turn out suitable for reductions, namely repeated reachability and boundedness [34] (these results are the core technical part); and when Monitor is the owner, we use two variants of universality of NTA_1 languages (these reductions are comparatively simpler).

Table 2 Summary of our undecidability results.

	Reactive synthesis	Church synthesis	
	any Agent	Agent = Timer	Agent = Monitor
Owner = Timer	$\mathcal{A} \in \mathrm{NTA}_1$ LCM repeated reach. (Theorem 7, Section 5)	$\mathcal{A} \in \mathrm{NTA}_1$ LCM repeated reach. (Theorem 7, Section 5)	$\mathcal{A} \in$ reach-$\mathrm{NTA}_1^{\mathrm{res}}$ LCM boundedness (Theorem 11, Section 6)
Owner = Monitor	$\mathcal{A} \in \mathrm{NTA}_1$ NTA_1 universality (Theorem 4, Section 3)	$\mathcal{A} \in \mathrm{NTA}_1$ NTA_1 sampled universality (Theorem 6, Section 3)	$\mathcal{A} \in \mathrm{NTA}_1$ NTA_1 universality (Theorem 4, Section 3)

The table specifies the class of winning conditions $\mathcal{A}$ for which we prove undecidability: $\mathcal{A} \in \mathrm{NTA}_1$, except for one exception where we need *1-resetting* reachability NTA_1, a slight extension of reachability NTA_1 that uses a limited form of ε-transitions that reset the clock whenever it reaches value 1. We believe that resorting to $\mathrm{NTA}_1^{\mathrm{res}}$ does not weaken our results, for the following two reasons: first, while NTA_1 with ε-transitions correspond to nondeterministic one-register automata (NRA_1) *with guessing* (see [20]), $\mathrm{NTA}_1^{\mathrm{res}}$ still correspond to NRA_1 *without guessing*; second, all our undecidability results translate from the timed setting to the data setting, where the winning sets are specified by NRA_1 without guessing.

3 Monitor is the owner

As a warm-up, we prove undecidability of both synthesis problems in the case where Monitor owns the winning condition, by reduction from two variants of the universality problem for NTA_1 (shown undecidable in [25] and [1]).

▶ **Theorem 4.** *When* Owner = Monitor*, the following problems are undecidable:*
1. *the timed reactive synthesis, irrespectively of* Agent*;*
2. *the timed Church synthesis, when* Agent = Monitor*.*

Proof. We reduce from the NTA_1 universality problem (does the language of a given $\mathcal{A} \in \mathrm{NTA}_1$ contain all infinite timed words?) [25]. Given an NTA_1 $\mathcal{A}$ over alphabet Σ, we construct a timed game where Monitor's alphabet is the singleton $\{\square\}$, Timer's alphabet is Σ,

and the winning condition of Monitor is $\mathcal{L}(\mathcal{A})$ (ignoring the letters "□"). Therefore, Timer wins if it produces a timed word over Σ that is not in $\mathcal{L}(\mathcal{A})$. Thus, Timer has a winning strategy if $\mathcal{L}(\mathcal{A})$ does not contain all timed words, and Monitor has a winning controller otherwise (the trivial memoryless one that keeps emitting "□"). ◀

▶ Remark 5. The reduction does not guarantee the existence of a controller for Timer, because Timer's strategy may need to produce an infinite timed word. However, in the case of reach-NTA_1 the strategy only needs to produce a finite prefix, and therefore we obtain HyperAckermann-hardness of both synthesis problems, regardless of Owner and Agent.

For undecidability of the timed Church synthesis problem when Agent = Timer, we turn to the *sampled* universality problem, a variant that assumes the *sampled* semantics $\mathcal{L}_\delta(\_)$ of timed automata. Given $\delta > 0$, we say that a timed word has *granularity* δ if all its timestamps are multiplicities of δ, and define $\mathcal{L}_\delta(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A})$ as the language of all timed words of granularity δ in $\mathcal{L}(\mathcal{A})$. Given an NTA $\mathcal{A}$, the *sampled* universality problem asks if for all $\delta > 0$, the language $\mathcal{L}_\delta(\mathcal{A})$ contains all timed words of granularity δ. The problem (called *universal sampled universality* in [1]) is known to be undecidable for NTA_1 [1, Thm. 3].

▶ **Theorem 6.** *The timed Church synthesis problem is undecidable when* Owner = Monitor *and* Agent = Timer.

Proof. We proceed similarly as in Theorem 4, but reduce from the NTA_1 sampled universality problem. Given an NTA_1 $\mathcal{A}$ over alphabet Σ, we construct the same timed game as in Theorem 4: $\text{M} = \{\square\}$, $\text{T} = \Sigma$, and the winning condition of Monitor is $\mathcal{L}(\mathcal{A})$ (ignoring "□").

If Timer has a winning controller $\mathcal{C}$ in this game, we take an arbitrary rational $\delta > 0$ such that all positive delays used by $\mathcal{C}$ are multiplicities of δ, and deduce that any infinite timed word output by $\mathcal{C}$ has granularity δ and is not in $\mathcal{L}_\delta(\mathcal{A})$.

In the opposite direction, suppose a timed word w of granularity δ is not accepted by $\mathcal{A}$. Assume, w.l.o.g., that $\mathcal{A}$ has a run labeled by w. We build a word w' whose run visits only finitely many different configurations. Let M be the maximal constant appearing in the guards of $\mathcal{A}$. If the valuations in the run labeled by w never exceed M, i.e. if $\mathcal{A}$ always resets the clock when it goes over M, then we take $w' = w$ and we are done because every valuation is always a multiplicity of δ and there are finitely many thereof between 0 and M. Otherwise, let $t_M > M$ be the first valuation in the run that exceeds M. We obtain w' by modifying w so that every time the valuation in the run is over M, it is exactly t_M (using repeating timestamps if necessary), but the run is unchanged when the valuation is at most M. Since valuations over M are indistinguishable by $\mathcal{A}$, w' is not accepted by $\mathcal{A}$. Therefore no accepting locations appear on the run from some point on, and hence the run necessarily forms some loop without accepting locations on it. We replace w' by the timed word w'' obtained by letting $\mathcal{A}$ repeat the loop infinitely. Like w', the word w'' has granularity δ and is not in $\mathcal{L}_\delta(\mathcal{A})$, but w'' is additionally ultimately periodic when seen as a sequence of letter-delay pairs $w'' = (a_1, \tau_1)(a_2, \tau_2)\cdots$. In consequence, Timer has a winning controller that produces w''. ◀

4 Lossy counter machines

We now introduce the model to be used in the forthcoming reductions. A *lossy counter machine* (LCM) is a tuple $\mathcal{M} = \langle \mathcal{C}, S, s_0, \mathcal{I} \rangle$, where $\mathcal{C}$ is a finite set of counters, S is a finite set of control locations, $s_0 \in S$ is the initial control location, and $\mathcal{I}$ is a finite set of instructions $I = (s, \text{op}, s')$, where $s, s' \in S$ and $\text{op} \in \{c{+}{+}, c{-}{-}, c{=}0? \mid c \in \mathcal{C}\}$ is the operation

of increment, decrement, or zero test on some counter $c \in \mathcal{C}$. We let $\mathcal{I}^c_{\text{inc}}, \mathcal{I}^c_{\text{dec}}, \mathcal{I}^c_{\text{zt}}$ denote the sets of instructions incrementing, decrementing, and zero testing counter c, respectively. We also let $\mathcal{I}^c = \mathcal{I}^c_{\text{inc}} \cup \mathcal{I}^c_{\text{dec}} \cup \mathcal{I}^c_{\text{zt}}$ be the set of all instructions affecting c.

A *configuration* of an LCM $\mathcal{M}$ is a pair $(s, \nu) \in S \times \mathbb{N}^{\mathcal{C}}$ where s is a control location, and ν is a counter valuation. Given counter valuations $\mu, \nu \in \mathbb{N}^{\mathcal{C}}$, we write $\mu \leq \nu$ whenever $\mu(c) \leq \nu(c)$ for every $c \in \mathcal{C}$. We define two equivalent semantics: the *lossy* semantics, and the *free-test* one (corresponding, intuitively, to postponing the losses until zero tests). A *run* of an LCM under either semantics is a maximal (finite or infinite) sequence of configurations starting from the initial configuration (s_0, ν_0), where $\nu_0 = 0^{\mathcal{C}}$, such that for every two consecutive configurations (s, ν) and (s', ν') there exists an instruction $I = (s, \text{op}, s') \in \mathcal{I}$ such that $(s, \nu) \xrightarrow{I} (s', \nu')$. Under the lossy semantics, $(s, \nu) \xrightarrow{I} (s', \nu')$ holds if

1. $I \in \mathcal{I}^c_{\text{inc}}$ and $\nu' \leq \nu[c{+}{+}]$, where $\nu[c{+}{+}]$ is as ν except that c is incremented by 1; or
2. $I \in \mathcal{I}^c_{\text{dec}}$ and $\nu' \leq \nu[c{-}{-}]$, where $\nu[c{-}{-}]$ is as ν except that c is decremented by 1; or
3. $I \in \mathcal{I}^c_{\text{zt}}$, $\nu(c) = 0$, and $\nu' \leq \nu$.

Under the free-test semantics, $(s, \nu) \xrightarrow{I} (s', \nu')$ holds if

1. $I \in \mathcal{I}^c_{\text{inc}}$ and $\nu' = \nu[c{+}{+}]$; or
2. $I \in \mathcal{I}^c_{\text{dec}}$ and $\nu' = \nu[c{-}{-}]$; or
3. $I \in \mathcal{I}^c_{\text{zt}}$ and $\nu' = \nu[c := 0]$ is as ν except that c is set to 0.

One can see the free-test semantics as delaying losses until the next zero test, at which point the counter loses its value entirely before proceeding to the next configuration.

We assume, w.l.o.g., that $\mathcal{M}$ is *deterministic*, i.e. that for any configuration (s, ν) there exists at most one instruction I leading to another configuration. Under the free-test semantics, this implies that a given configuration only has at most one successor. Under the lossy semantics, there may be a number of successor configurations differing only on the counter valuations (how much is lost for each counter after the instruction). Let $\text{Runs}_{\mathcal{M}}$ denote the runs of $\mathcal{M}$. We assume, w.l.o.g., that they are all infinite.

The *boundedness* problem asks whether all runs of a given LCM $\mathcal{M}$ visit only finitely many configurations, or equivalently whether all runs of $\mathcal{M}$ are bounded in their valuations. The *repeated reachability problem* asks if $\mathcal{M}$ has an infinite run visiting infinitely often a given control location $s \in S$. Both problems are known to be undecidable already for 4 counters [29, Thms. 10,12] (an excellent survey is [34]), and the choice of semantics is irrelevant.

5 Timer is both the owner and the agent

In this section we prove undecidability of both synthesis problems when Timer owns the winning condition, except when Agent = Monitor (investigated in the next section).

▶ **Theorem 7.** *When* Owner = Timer*, the following problems are undecidable:*

- *the timed reactive synthesis, irrespectively of* Agent*;*
- *the timed Church synthesis, when* Agent = Timer*.*

(Note the symmetry of Theorems 4 and 7 along the exchange of roles of Timer and Monitor.)

In the rest of this section we prove Theorem 7 by providing a reduction from the repeated reachability problem for LCM. To this aim we fix a lossy counter machine $\mathcal{M} = \langle \mathcal{C}, S, s_0, \mathcal{I} \rangle$ with four counters $\mathcal{C} = \{c_1, c_2, c_3, c_4\}$ and a location $s \in S$, and construct a timed game with the property that Timer has a winning controller if $\mathcal{M}$ repeatedly reaches s, and Monitor has a winning strategy otherwise (see Lemma 10). In the proof we assume lossy semantics of $\mathcal{M}$. Our approach is inspired by the proof of [32, Thm. 8.4] (we note however substantial

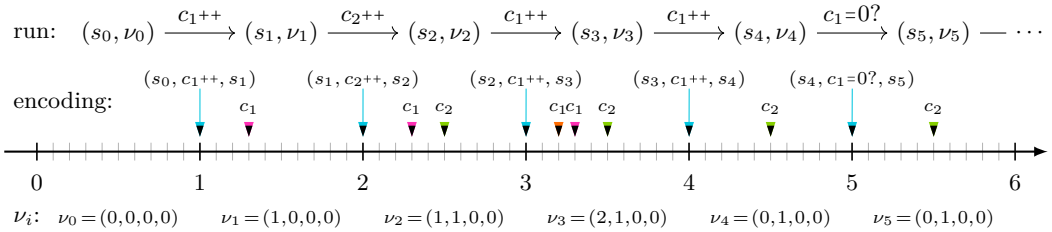

Figure 1 Illustration of the encoding of runs used in this section.

differences: timed Church synthesis was considered there with Timer's winning condition specified by NTA_2 while we restrict ourselves to NTA_1, and moreover, there Monitor's controller was sought, while we seek Timer's one).

The idea of reduction. In the course of the game, Timer is tasked with producing an increasingly longer timed word supposed to be an encoding of a run of $\mathcal{M}$. However, Timer may also *cheat* and produce a timed word which contains an *error* and therefore is not a correct run encoding. We distinguish four types of errors. In order to prevent Timer from cheating, Monitor verifies if the encoding proposed by Timer is correct, and if it is not, Monitor has to declare the detected type of error correctly, and immediately, that is in the same round the error occurs. Therefore one way of winning by Timer is to see s infinitely often while seeing no error declared by Monitor (using cheating or not); or to mislead Monitor about the correctness of encoding by cheating and either seeing no immediate error declaration, or seeing an error declaration of wrong type. On the other hand, when Monitor manages to declare immediately the correct type of error, the play is winning for it irrespectively of the continuation.

Encoding of runs of $\mathcal{M}$. A central ingredient of our reduction is the encoding of runs of $\mathcal{M}$ as timed words. Let $\Sigma_{\mathcal{M}} = \mathcal{C} \cup \mathcal{I}$. We first introduce the *valuation encoding*. For a valuation $\nu = (v_1, v_2, v_3, v_4) \in \mathbb{N}^{\mathcal{C}}$ define $\mathrm{enc}(\nu) = c_1^{v_1} c_2^{v_2} c_3^{v_3} c_4^{v_4}$, a finite untimed word consisting of four *segments*. The set of all such encodings is $\mathrm{ValEnc}_{\mathcal{M}} = c_1^* c_2^* c_3^* c_4^*$.

Next, we define the *run encoding*. The function $\mathrm{enc}\colon \mathrm{Runs}_{\mathcal{M}} \to \Sigma_{\mathcal{M}}^{\omega}$ maps a run $\rho = (s_0,\nu_0) \xrightarrow{I_1} (s_1,\nu_1) \xrightarrow{I_2} \cdots$ to the infinite untimed word $\mathrm{enc}(\rho) = \mathrm{enc}(\nu_0)\, I_1\, \mathrm{enc}(\nu_1)\, I_2 \cdots$. Let $\mathrm{RunEnc}_{\mathcal{M}} = \{\mathrm{enc}(\rho) \mid \rho \in \mathrm{Runs}_{\mathcal{M}}\}$ be the set of valid untimed encodings.

Intuitively, we exploit the timed structure of a word to enforce the correctness of run encodings (see Lemma 8). To this end, we define a timed language $\mathrm{RunEnc}_{\mathcal{M}}^{\mathbb{T}}$. We specify it using an untimed ω-regular language $\mathrm{Reg}_{\mathcal{M}}$ which enforces the structural shape of runs of $\mathcal{M}$, and timed languages $A_{\mathcal{M}}^{\mathbb{T}}$, $B_{\mathcal{M}}^{\mathbb{T}}$ and $C_{\mathcal{M}}^{\mathbb{T}}$, which impose additional timed properties. Let $\mathrm{Reg}_{\mathcal{M}}$ consist of all (untimed) words $w \in \Sigma_{\mathcal{M}}^{\omega}$ that satisfy the following regular conditions:

Block structure: w is an infinite interleaving of valuation encodings and instructions, i.e., w is of the form $w = \mathrm{enc}(\nu_0)\, I_1\, \mathrm{enc}(\nu_1)\, I_2 \cdots \in (\mathrm{ValEnc}_{\mathcal{M}}\, \mathcal{I})^{\omega}$ with I_1 starting from s_0.

Instruction compatibility: Each instruction's target state is the source state of the next one, i.e., for every infix $I_i\, \mathcal{C}^*\, I_{i+1}$ we have $I_i = (\_, \_, s)$ and $I_{i+1} = (s, \_, \_)$ for some s.

Consistency with zero tests: every infix $I_i\, \mathrm{enc}(\nu_i)\, I_{i+1}$ of w with $I_{i+1} \in \mathcal{I}_{\mathrm{zt}}^{c}$ verifies $\mathrm{enc}(\nu_i) \in (\mathcal{C} \setminus \{c\})^*$, i.e., the symbol c does not appear in $\mathrm{enc}(\nu_i)$.

Clearly $\mathrm{Reg}_{\mathcal{M}} \supseteq \mathrm{RunEnc}_{\mathcal{M}}$. Let $\mathrm{Reg}^{\mathbb{T}}_{\mathcal{M}} = \mathrm{untime}^{-1}(\mathrm{Reg}_{\mathcal{M}})$ contain all timed words whose untiming is in $\mathrm{Reg}_{\mathcal{M}}$. We define $\mathrm{RunEnc}^{\mathbb{T}}_{\mathcal{M}}$ as the intersection of four timed languages:

$$\mathrm{RunEnc}^{\mathbb{T}}_{\mathcal{M}} = \mathrm{Reg}^{\mathbb{T}}_{\mathcal{M}} \cap A^{\mathbb{T}}_{\mathcal{M}} \cap B^{\mathbb{T}}_{\mathcal{M}} \cap C^{\mathbb{T}}_{\mathcal{M}} \subseteq (\Sigma_{\mathcal{M}} \times \mathbb{Q}_{\geq 0})^{\omega}$$

where $A^{\mathbb{T}}_{\mathcal{M}}$, $B^{\mathbb{T}}_{\mathcal{M}}$ and $C^{\mathbb{T}}_{\mathcal{M}}$ are languages of infinite timed words w satisfying the conditions given below. For simplicity, we define the condition for $C^{\mathbb{T}}_{\mathcal{M}}$ (the most intricate), assuming that $w \in L = \mathrm{Reg}^{\mathbb{T}}_{\mathcal{M}} \cap A^{\mathbb{T}}_{\mathcal{M}} \cap B^{\mathbb{T}}_{\mathcal{M}}$, as it is irrelevant how it treats words outside L.

Strict monotonicity ($A^{\mathbb{T}}_{\mathcal{M}}$): w is strictly monotonic (no timestamp repeats).

Blocks align to unit intervals ($B^{\mathbb{T}}_{\mathcal{M}}$): symbols from $\mathcal{I}$ appear in w with consecutive integer timestamps starting from 1.

Well-alignment of valuations encodings ($C^{\mathbb{T}}_{\mathcal{M}}$): every maximal infix of w of the form $\mathrm{enc}(\nu_i)\ \ I_i\,\mathrm{enc}(\nu_{i+1})$ verifies the following conditions, for all counters $c \in \mathcal{C}$:

1. if $I_i \notin \mathcal{I}^c_{\mathrm{dec}} \cup \mathcal{I}^c_{\mathrm{inc}}$ then $\nu_{i+1}(c) \leq \nu_i(c)$, and a timed letter (c,t) appears in $\mathrm{enc}(\nu_{i+1})$ only if $(c, t-1)$ appears in $\mathrm{enc}(\nu_i)$;
2. if $I_i \in \mathcal{I}^c_{\mathrm{inc}}$, then $\nu_{i+1}(c) \leq \nu_i(c) + 1$, and the encoding verifies the case 1 except for a (potential) one extra letter c appearing in the beginning of the c-segment in $\mathrm{enc}(\nu_{i+1})$ within less than one unit of time from the first symbol c of $\mathrm{enc}(\nu_i)$.
3. if $I_i \in \mathcal{I}^c_{\mathrm{dec}}$, then $\nu_{i+1}(c) \leq \nu_i(c) - 1$, and the encoding verifies the case 1 except for the first letter c appearing in $\mathrm{enc}(\nu_i)$ which can not appear in $\mathrm{enc}(\nu_{i+1})$.

According to the definition of $C^{\mathbb{T}}_{\mathcal{M}}$, every increment (resp. decrement) of a counter c results in adding (resp. removing) one letter c in the *beginning* of the c-segment. An illustration is provided in Figure 1.

▶ **Lemma 8.** $\mathrm{RunEnc}_{\mathcal{M}} = \{\mathrm{untime}(w) \mid w \in \mathrm{RunEnc}^{\mathbb{T}}_{\mathcal{M}}\}$.

The timed game. We define the timed synthesis game where Timer's actions are $\mathrm{T} = \Sigma_{\mathcal{M}}$ and Monitor's actions are $\mathrm{M} = \{\nu, \chi_{\mathrm{R}}, \chi_{\mathrm{A}}, \chi_{\mathrm{B}}, \chi_{\mathrm{C}}\}$. For the definition of the winning condition it is crucial that all the languages $\mathrm{Reg}^{\mathbb{T}}_{\mathcal{M}}, A^{\mathbb{T}}_{\mathcal{M}}, B^{\mathbb{T}}_{\mathcal{M}}$ and $C^{\mathbb{T}}_{\mathcal{M}}$ can be defined *locally*: an infinite word w belongs to $\mathrm{Reg}^{\mathbb{T}}_{\mathcal{M}}$ exactly when all (finite) prefixes of w belong to a certain local language $\mathrm{Reg}^{\mathbb{T},\ell}_{\mathcal{M}} \subseteq (\Sigma_{\mathcal{M}} \times \mathbb{Q}_{\geq 0})^*$ of *finite* timed words, which is moreover recognised by a reach-NTA$_1$; and likewise for the $A^{\mathbb{T}}_{\mathcal{M}}$, $B^{\mathbb{T}}_{\mathcal{M}}$ and $C^{\mathbb{T}}_{\mathcal{M}}$. Specifically:

- $\mathrm{Reg}^{\mathbb{T},\ell}_{\mathcal{M}}$: the finite prefix satisfies the defining conditions of $\mathrm{Reg}_{\mathcal{M}}$.
- $A^{\mathbb{T},\ell}_{\mathcal{M}}$: the last two timestamps are nonequal (or the word is of length 1, the border case).
- $B^{\mathbb{T},\ell}_{\mathcal{M}}$: if the last letter is from $\mathcal{I}$, then no $\mathcal{I}$ appears in the last open unit interval, and $\mathcal{I}$ appears exactly one time unit before the last timestamp; and if the last letter is not from $\mathcal{I}$, some $\mathcal{I}$ occurs less that one unit before the last timestamp. (We omit the border case.)
- $C^{\mathbb{T},\ell}_{\mathcal{M}}$: the last timed letter of the word, if it is $(c,t) \in \mathcal{C} \times \mathbb{Q}_{\geq 0}$, appears also one unit of time before as $(c, t-1)$, except when it is the first letter in the c-segment and the last instruction is an increment of c. Moreover, when (c,t) is the first letter in the c-segment and the last instruction is a decrement of c, the language $C^{\mathbb{T},\ell}_{\mathcal{M}}$ requires that the first letter in the previous c-segment was removed, i.e., its timestamp is strictly smaller that $t-1$. As before, we conveniently assume that the word is in $L = \mathrm{Reg}^{\mathbb{T},\ell}_{\mathcal{M}} \cap A^{\mathbb{T},\ell}_{\mathcal{M}} \cap B^{\mathbb{T},\ell}_{\mathcal{M}}$, as it is irrelevant which words from $\widehat{L} = \widehat{\mathrm{Reg}^{\mathbb{T},\ell}_{\mathcal{M}}} \cup \widehat{A^{\mathbb{T},\ell}_{\mathcal{M}}} \cup \widehat{B^{\mathbb{T},\ell}_{\mathcal{M}}}$ belong to $C^{\mathbb{T},\ell}_{\mathcal{M}}$.

Note that $C^{\mathbb{T},\ell}_{\mathcal{M}}$ is the most difficult one, and the only one for which we will need non-determinism.

The winning condition is also defined mostly locally, by a combination of restrictions imposed on Timer's or Monitor's moves. To this aim we use the projections of finite words, $\mathrm{proj}_{\mathrm{M}} \colon (\mathrm{T} \times \mathrm{M} \times \mathbb{Q}_{\geq 0})^* \to \mathrm{M}^*$ and $\mathrm{proj}_{\mathrm{T},\mathbb{T}} \colon (\mathrm{T} \times \mathrm{M} \times \mathbb{Q}_{\geq 0})^* \to (\mathrm{T} \times \mathbb{Q}_{\geq 0})^*$, as well as their inverses $\mathrm{proj}_{\mathrm{M}}^{-1}$ and $\mathrm{proj}_{\mathrm{T},\mathbb{T}}^{-1}$. We define Timer's winning set as

$$W_{\mathcal{M}} = \mathrm{Reach}(V_{\mathcal{M}}^{\mathbb{T}}) \cup \Big(\mathrm{proj}_{\mathrm{T},\mathbb{T}}^{-1}(\mathrm{Inf}(s)) \cap \mathrm{proj}_{\mathrm{M}}^{-1}(\{\checkmark\}^{\omega}) \Big)$$

where $\mathrm{Inf}(s)$ is the (untimed) regular language "the location s appears infinitely often", $\mathrm{Reach}(V_{\mathcal{M}}^{\mathbb{T}})$ stands for the language of those infinite timed words which have some prefix in $V_{\mathcal{M}}^{\mathbb{T}}$, and $V_{\mathcal{M}}^{\mathbb{T}}$ itself is the following union of languages of finite timed words:

$$\begin{aligned}
V_{\mathcal{M}}^{\mathbb{T}} = {} & \big(\mathrm{proj}_{\mathrm{M}}^{-1}(\checkmark^* \chi_{\mathrm{R}}) \cap \mathrm{proj}_{\mathrm{T},\mathbb{T}}^{-1}(\mathrm{Reg}_{\mathcal{M}}^{\mathbb{T},\ell})\big) \ \cup && \text{(Monitor wrongly claims an Reg error)} \\
& \big(\mathrm{proj}_{\mathrm{M}}^{-1}(\checkmark^* \chi_{\mathrm{A}}) \cap \mathrm{proj}_{\mathrm{T},\mathbb{T}}^{-1}(A_{\mathcal{M}}^{\mathbb{T},\ell})\big) \ \cup && \text{(Monitor wrongly claims an } A \text{ error)} \\
& \big(\mathrm{proj}_{\mathrm{M}}^{-1}(\checkmark^* \chi_{\mathrm{B}}) \cap \mathrm{proj}_{\mathrm{T},\mathbb{T}}^{-1}(B_{\mathcal{M}}^{\mathbb{T},\ell})\big) \ \cup && \text{(Monitor wrongly claims a } B \text{ error)} \\
& \big(\mathrm{proj}_{\mathrm{M}}^{-1}(\checkmark^* \chi_{\mathrm{C}}) \cap \mathrm{proj}_{\mathrm{T},\mathbb{T}}^{-1}(C_{\mathcal{M}}^{\mathbb{T},\ell} \cup \widehat{\mathrm{Reg}_{\mathcal{M}}^{\mathbb{T},\ell}} \cup \widehat{A_{\mathcal{M}}^{\mathbb{T},\ell}} \cup \widehat{B_{\mathcal{M}}^{\mathbb{T},\ell}})\big) \\
& && \text{(Monitor wrongly claims a } C \text{ error)}
\end{aligned}$$

Thus, Timer wins if s is occurring infinitely often and Monitor declares no error, i.e., plays exclusively $\checkmark$ moves, or some finite prefix of play is in $V_{\mathcal{M}}^{\mathbb{T}}$, namely Monitor declares a wrong type of error. Specifically, Monitor declares χ_{R}, χ_{A} or χ_{B} in the round when the corresponding local condition holds, or χ_{C} in the round when either the local condition $C_{\mathcal{M}}^{\mathbb{T},\ell}$ holds, or some of the other three local conditions fails. Intuitively, χ_{R}, χ_{A} or χ_{B} are prioritised over χ_{C}: in order to win by declaring an error, Monitor must declare χ_{R}, χ_{A} or χ_{B} if the corresponding local condition fails, and may only declare χ_{C} otherwise.

▶ **Lemma 9.** *$W_{\mathcal{M}}$ is recognised by an NTA_1.*

Correctness of reduction. We prove the following lemma:

▶ **Lemma 10.** Timer *has a winning controller if $\mathcal{M}$ repeatedly reaches s, and* Monitor *has a winning strategy otherwise.*

Proof. Due to a well-quasi-order on configurations and the lossy semantics, if $\mathcal{M}$ repeatedly reaches s then $\mathcal{M}$ has a *lasso* run repeatedly reaching s [34]: a run that is cyclic from some point on. The run is thus bounded by some $k \in \mathbb{N}$, and Timer has a winning controller that produces an encoding of this run of granularity at most $\delta = 1/4(k+1)$. Indeed, with this granularity, Timer can encode all potential valuations by using k "slots" in an interval of length $1/4$ for each of the segments.

Conversely, if $\mathcal{M}$ does not repeatedly reach s then whenever Timer produces a correct run it necessarily visits s finitely often. Then Monitor has a winning strategy that records all the history and keeps playing $\checkmark$ as long as Timer is not cheating, and is able to detect all kinds of errors produced by Timer in case Timer cheats. ◀

6 Timer is the owner but not the agent

In this section we prove undecidability in the last remaining case:

▶ **Theorem 11.** *The timed Church synthesis problem is undecidable when* Owner = Timer *and* Agent = Monitor.

(Note the similarity of Theorems 6 and 11 along the exchange of roles of Timer and Monitor.)

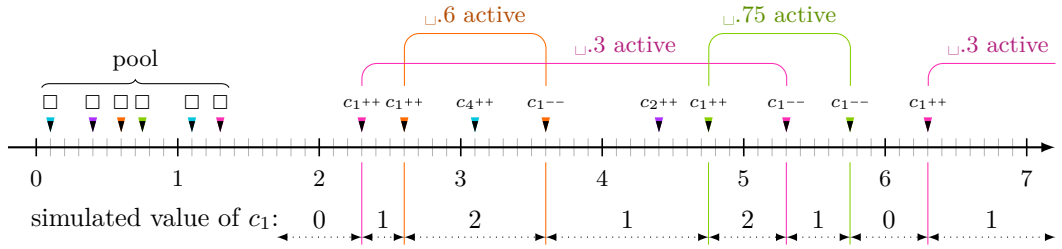

Figure 2 Example encoding of a run. For each instruction (s, op, s'), only op is shown.

We reduce from the LCM boundedness problem, assuming free-test semantics. Let us fix a 4-counter LCM $\mathcal{M} = \langle \mathcal{C}, S, s_0, \mathcal{I} \rangle$, where $\mathcal{C} = \{c_1, c_2, c_3, c_4\}$. Let $\mathrm{T} = \mathcal{I} \cup \{\square\}$ and $\mathrm{M} = \{\checkmark, \lightning\}$. We define a timed game with Timer's winning condition $W_{\mathcal{M}}$ such that Monitor has a winning controller if and only if $\mathcal{M}$ is bounded.

1-resetting NTA_1. In this case, we need a slight extension of NTA_1 to define winning conditions and controllers: NTA with 1 clock and very limited form of ε-transitions that reset the clock every time it equals 1. Let $\mathbb{F} = \mathbb{Q}_{\geq 0} \cap [0, 1)$ denote the set of fractional parts. For any time value $t \in \mathbb{R}_{\geq 0}$, let $\mathrm{frac}(t)$ denote its fractional part, i.e., the unique value in $\mathbb{F}$ such that $t = n + \mathrm{frac}(t)$ for some $n \in \mathbb{N}$. If ν is a valuation, $\mathrm{frac}(\nu)$ denotes the valuation $\{x \mapsto \mathrm{frac}(\nu(x))\}$.

A *1-resetting NTA*$_1$ ($\mathrm{NTA}_1^{\mathrm{res}}$) is an NTA_1 as defined in Section 2 with only the following modification: $(\ell, \nu) \xrightarrow{\delta, \tau} (\ell', \nu')$ (with $a \in \Sigma$) if and only if $\delta = (\ell, a, g, Y, \ell') \in \Delta$ is a transition of $\mathcal{A}$ such that $\mathrm{frac}(\nu + \tau) \models g$, and $\nu' = \mathrm{frac}(\nu + \tau)[Y := 0]$. The 1-resetting NTA_1 can be simulated using NTA_1 with ε-transitions: in every location $\ell \in L$ add a self-loop $(\ell, \varepsilon, x = 1, \{x\}, \ell)$.

The idea of reduction. Intuitively speaking, Timer is tasked with simulating a run of $\mathcal{M}$, and Monitor can point out when they think that Timer made a mistake in the simulation. More specifically, Timer will play instructions of $\mathcal{M}$, and the time values will be used to encode the valuations of counters, as described below (the encoding is different from the one in the previous section). After each move by Timer, Monitor can either say that the simulation is correct so far, or that Timer made a mistake in their last move. In the former case, if Timer actually made a mistake then Timer wins (Monitor must point out any mistake), and if there was no mistake then the game continues (if the game continues like this forever, Monitor wins). In the latter case, the game is essentially immediately over: either Monitor is right and Timer made a mistake, which will be winning for Monitor, or Monitor is wrong and Timer's last move was correct, in which case Timer will be winning.

The way time is used to encode counter valuations is the following. The fractional part of a timestamp acts as an identifier which allows an increment to be later matched with a corresponding decrement. For instance, an increment at time 2.314 can be later matched with a decrement at time 7.314. With this, the valuation of a counter is simply the number of increments/fractional parts that have not been matched with a later decrement with the same fractional part. See Figure 2 for an illustration.

An obvious implementation of this idea would require, in the case where a fractional part is used for the first time ever in an incrementing instruction, checking that it is fresh using unrestricted "guessing" ε-transition (i.e., guess the fractional part before the first instruction, then check it never appears until the very last position). To avoid this, we introduce a

pool of fractional parts at the start of the run encoding, denoted by the symbol "□". The idea is that Timer must include at the beginning a number of relevant fractional parts that will be used later in the run encoding. Then, we can eliminate time guessing by instead non-deterministically choosing a *position* in the pool, resetting the clock at that point and then every time it equals 1. If the last position in the word is the only position seen with the clock exactly at zero, then we know this fractional part was never used before.

The encoding of runs of $\mathcal{M}$. We define a timed word encoding of runs of $\mathcal{M}$ in three parts. First, we capture the regular properties of their projection onto T. Let $\mathrm{proj}_{\mathrm{T}} \colon (\mathrm{T} \times \mathrm{M} \times \mathbb{Q}_{\geq 0})^* \to \mathrm{T}^*$ be the natural projection. Define $\mathrm{Reg}_{\mathcal{M}} \subseteq (\mathrm{T} \times \mathrm{M} \times \mathbb{Q}_{\geq 0})^*$ as the set of all finite timed words w such that $\mathrm{proj}_{\mathrm{T}}(w) = \square^k I_1 I_2 \cdots I_n$, instruction I_1 starts from the initial location, and each I_i is compatible with I_{i+1} in the sense defined earlier.

It remains to state the role of timestamps, on positions with a symbol in $\mathcal{I}$, in maintaining the valuations of $\mathcal{M}$. Consider a finite timed word $w \in (\mathrm{T} \times \mathrm{M} \times \mathbb{Q}_{\geq 0})^*$. We say that a fractional part $f \in \mathbb{F}$ is *active* for counter c in w if the last timed letter (I, t) with $I \in \mathcal{I}^c$ and $\mathrm{frac}(t) = f$ increments c ($I \in \mathcal{I}^c_{\mathrm{inc}}$) and appears after the last zero-test of c. Conversely, f is *inactive* for c in w if since the last zero-test of c there is no occurrence of f with an instruction $I \in \mathcal{I}^c$ involving c or if the last such occurrence is in $\mathcal{I}^c_{\mathrm{dec}}$. For any prefix w of a run encoding, the corresponding valuation of counter c is the number of distinct fractional parts that are active for c. Let $\mathrm{val}(w)$ denote this valuation.

Recall the form of runs of $\mathcal{M}$: $\rho = (s_0, \nu_0) \xrightarrow{I_1} (s_1, \nu_1) \xrightarrow{I_2} \cdots$, as an alternating sequence of configurations and instructions, with $\nu_0 = 0^{\mathcal{C}}$. We inductively define $\rho(w)$, a finite alternating sequence of configurations and instructions associated with any $w \in \mathrm{Reg}_{\mathcal{M}}$:

$$\rho(w) = \begin{cases} (s_0, \nu_0) & \text{if } \mathrm{proj}_{\mathrm{T}}(w) \in \square^*, \\ \rho(w') \xrightarrow{I} (s, \mathrm{val}(w)) & \text{if } \mathrm{proj}_{\mathrm{T}}(w) = w' \cdot I \text{ and } I = (s', \mathrm{op}, s). \end{cases}$$

Note that $\rho(w)$ need not be a run of $\mathcal{M}$ for two reasons. First, valuations need not be correlated with the instructions (e.g., a run may feature an incrementing instruction while the corresponding valuation remains unchanged or decreases). Second, a decrementing instruction may occur even when the valuation of the corresponding counter is zero. Conversely, if valuations are updated correctly and decrementing instructions occur only when the valuation is non-zero, then $\mathrm{Reg}_{\mathcal{M}}$ ensures that $\rho(w)$ is a valid run of $\mathcal{M}$. To enforce validity, we introduce rules that must be satisfied by the timestamp t of every instruction I occurring in w:

Rule 1: If $I \in \mathcal{I}^c_{\mathrm{inc}}$, then the fractional part of t must be inactive for c.
Rule 2: If $I \in \mathcal{I}^c_{\mathrm{dec}}$, then the fractional part of t must be active for c.

It is easy to see that following both rules guarantees maintaining the correct counter valuations. Moreover, it prevents one from adding a decrementing instruction when the counter valuation is zero, as there would be no active fractional part to pick. Note that there are no constraints on zero-test instructions, their timestamps are irrelevant.

▷ Claim 12. If all instructions in $w \in \mathrm{Reg}_{\mathcal{M}}$ satisfy Rules 1 and 2, then $\rho(w)$ is a run of $\mathcal{M}$.

The timed game. For each of the two rules, we now define a language of finite timed words that requires that the rule is satisfied by the last letter, and another one that requires that the rule is broken. Let $\mathrm{Err}^{\mathrm{R1}}_{\mathcal{M}}$ be the language of finite timed words where the last letter is an increment instruction that breaks Rule 1. Moreover, let $\mathrm{Ok}^{\mathrm{R1}}_{\mathcal{M}}$ be the language of words where the last letter satisfies Rule 1 and *the last fractional part is in the pool*. Note that any

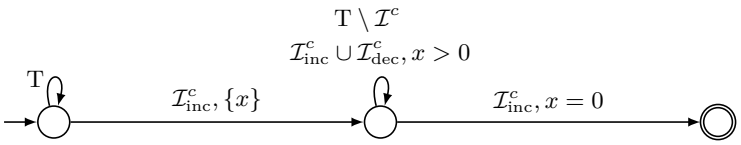

Figure 3 An $\text{NTA}_1^{\text{res}}$ for $\text{Err}_{\mathcal{M}}^{\text{R1}}$ comprises four copies of the above automaton, one for each $c \in \mathcal{C}$.

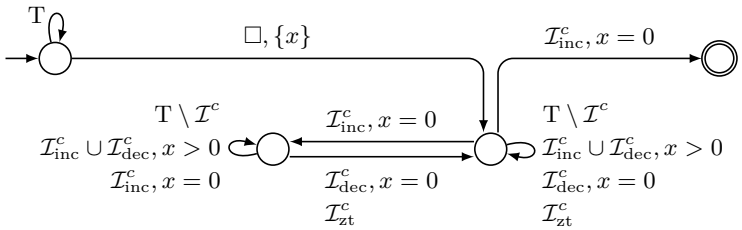

Figure 4 An $\text{NTA}_1^{\text{res}}$ recognising $\text{Ok}_{\mathcal{M}}^{\text{R1}}$ consists of four copies of the automaton above, one for each $c \in \mathcal{C}$, and an additional branch that checks that the last transition is in $\mathcal{I} \setminus \mathcal{I}_{\text{inc}}$ (omitted).

timed word ending in either a decrementing or zero-test instruction is in $\text{Ok}_{\mathcal{M}}^{\text{R1}}$, and is not in $\text{Err}_{\mathcal{M}}^{\text{R1}}$. $\text{Err}_{\mathcal{M}}^{\text{R1}}$ is recognised by the 1-resetting reachability NTA_1 given in Figure 3, while $\text{Ok}_{\mathcal{M}}^{\text{R1}}$ is recognised by the 1-resetting reachability NTA_1 given in Figure 4.

The languages $\text{Err}_{\mathcal{M}}^{\text{R1}}$ and $\text{Ok}_{\mathcal{M}}^{\text{R1}}$ are readily seen to be disjoint. Note however that the languages are not complements. Indeed, a word that satisfies Rule 1 but whose last fractional part does not appear in the pool at the beginning belongs to neither language.

The definitions and automata for $\text{Err}_{\mathcal{M}}^{\text{R2}}$ and $\text{Ok}_{\mathcal{M}}^{\text{R2}}$ mirror those of $\text{Ok}_{\mathcal{M}}^{\text{R1}}$ and $\text{Err}_{\mathcal{M}}^{\text{R1}}$ respectively, with $\text{Err}_{\mathcal{M}}^{\text{R2}}$ being the language of words breaking Rule 2 where *the last fractional part is in the pool*, and $\text{Ok}_{\mathcal{M}}^{\text{R2}}$ being the language of words where the last letter satisfies Rule 2. Again, a word not ending in a decrement is automatically in $\text{Ok}_{\mathcal{M}}^{\text{R2}}$ and not in $\text{Err}_{\mathcal{M}}^{\text{R2}}$.

Let $\text{Err}_{\mathcal{M}} = \text{Reg}_{\mathcal{M}} \cap \left(\text{Err}_{\mathcal{M}}^{\text{R1}} \cup \text{Err}_{\mathcal{M}}^{\text{R2}}\right)$ and $\text{Ok}_{\mathcal{M}} = \text{Reg}_{\mathcal{M}} \cap \text{Ok}_{\mathcal{M}}^{\text{R1}} \cap \text{Ok}_{\mathcal{M}}^{\text{R2}}$. We define Timer's winning condition as the following language of infinite timed words:

$$W_{\mathcal{M}} = \text{Reach}\left\{w \mid (\text{proj}_{\text{M}}(w) \in \checkmark^* \chi \wedge w \in \text{Ok}_{\mathcal{M}}) \vee (\text{proj}_{\text{M}}(w) \in \checkmark^* \checkmark \wedge w \in \text{Err}_{\mathcal{M}})\right\}.$$

Intuitively, Timer wins if at any point Monitor makes a mistake in its claim by playing χ when the sequence given by Timer is actually correct or playing $\checkmark$ when there is an error. However the play continues after this point, Timer will be winning. On the other hand, Monitor wins if either it plays χ at a point in which the play is not in $\text{Ok}_{\mathcal{M}}$, and then any continuation will be outside of $W_{\mathcal{M}}$, or if it plays $\checkmark$ forever while the play is never in $\text{Err}_{\mathcal{M}}$. Note that the case where Timer plays outside of $\text{Reg}_{\mathcal{M}}$ is covered both by not being in $\text{Ok}_{\mathcal{M}}$, so Monitor can play χ if this happens and win, or by not being in $\text{Err}_{\mathcal{M}}$, so Monitor can play $\checkmark$ if this happens and win as well.

▶ **Lemma 13.** *$W_{\mathcal{M}}$ is recognised by a reach-NTA_1^{res}.*

Proof. Both $\text{Err}_{\mathcal{M}}$ and $\text{Ok}_{\mathcal{M}}$ are recognisable by 1-resetting reachability NTA_1s. For $\text{Err}_{\mathcal{M}}$ this follows from the closure of reach-$\text{NTA}_1^{\text{res}}$ under union of languages, and intersection with $\text{Reg}_{\mathcal{M}}$ does not add clocks. As for $\text{Ok}_{\mathcal{M}}$, although intersection typically adds up the numbers of clocks, we can branch based on the final letter to simulate either $\text{Ok}_{\mathcal{M}}^{\text{R1}}$ or $\text{Ok}_{\mathcal{M}}^{\text{R2}}$, requiring only one clock. ◀

Correctness. Before stating correctness of the reduction, let us give some intuition about the pool at the beginning of the play. It is in Timer's best interest to accurately list as many fractional parts that will be used later, but only finitely many. The finite part is obvious: if Timer only plays $\square$ forever then Monitor just plays $\checkmark$ forever and wins, so Timer must start the real run encoding at some point. The incentive for Timer to play many fractional parts comes from its winning condition: it wins if Monitor makes a mistake. For example, if Timer plays an increment transition with some inactive fractional part f and Monitor answers with χ, Timer wins iff the play is in $\mathrm{Ok}_{\mathcal{M}}^{\mathrm{R1}}$. But $\mathrm{Ok}_{\mathcal{M}}^{\mathrm{R1}}$ accepts only if the last fractional part occurs in the pool. So if f is not in the pool, the play is not in $\mathrm{Ok}_{\mathcal{M}}$ (despite being a correct encoding of a run), and Monitor wins by playing χ. On the other hand, there is no possible disadvantage to adding more fractional parts to the pool, even some that will never be used later, and so this is Timer's incentive for filling the pool as much as possible.

▶ **Lemma 14.** *Let $w \in \mathrm{Reg}_{\mathcal{M}}$. We have the following implications:*

1. $\rho(w) \in \mathrm{Runs}_{\mathcal{M}} \Rightarrow$ *all prefixes w' of w satisfy $w' \notin \mathrm{Err}_{\mathcal{M}}$.*
2. $\rho(w) \notin \mathrm{Runs}_{\mathcal{M}} \Rightarrow$ *some prefix $w' = w'' \cdot (I, m, t)$ of w satisfies $\rho(w'') \in \mathrm{Runs}_{\mathcal{M}}$, $\rho(w') \notin \mathrm{Runs}_{\mathcal{M}}$, and $w' \notin \mathrm{Ok}_{\mathcal{M}}$.*

Proof.

1. By induction on prefixes of w. The base case, for prefixes where Timer has only played $\square$ letters so far, is trivial. Let w' be a prefix of w. Since $\mathrm{Runs}_{\mathcal{M}}$ is prefix-closed and $\rho(w')$ is a prefix of $\rho(w)$, $\rho(w') \in \mathrm{Runs}_{\mathcal{M}}$. Assume by induction hypothesis that all strict prefixes of w' are not in $\mathrm{Err}_{\mathcal{M}}$. Let $w' = w'' \cdot (I, m, t)$ with $I \in \mathcal{I}$, $m \in \mathrm{M}$, and let $f = \mathrm{frac}(t) \in \mathbb{F}$. We look at the different cases for I to show that $w' \notin \mathrm{Err}_{\mathcal{M}}$.
   - Suppose $I \in \mathcal{I}_{\mathrm{inc}}^c$. We have $\rho(w') \in \mathrm{Runs}_{\mathcal{M}}$, $\rho(w'') \in \mathrm{Runs}_{\mathcal{M}}$, and $\rho(w') = \rho(w'') \xrightarrow{I} (s, \mathrm{val}(w'))$. Thus $\mathrm{val}(w') = \mathrm{val}(w'')[c{+}{+}]$, and from this we have that f is inactive for c in w''. This does not mean that $w' \in \mathrm{Ok}_{\mathcal{M}}^{\mathrm{R1}}$ as f could be missing from the pool, but it does mean that $w' \notin \mathrm{Err}_{\mathcal{M}}^{\mathrm{R1}}$ as $\mathrm{Err}_{\mathcal{M}}^{\mathrm{R1}}$ can only accept if the last fractional part is already active. Thus $w' \notin \mathrm{Err}_{\mathcal{M}}$.
   - Suppose $I \in \mathcal{I}_{\mathrm{dec}}^c$. Since the run is correct, we have that $\mathrm{val}(w'')(c) > 0$ and $\mathrm{val}(w') = \mathrm{val}(w'')[c{-}{-}]$, so f must be active for c in w''. Then $w' \in \mathrm{Ok}_{\mathcal{M}}^{\mathrm{R2}}$ which implies $w' \notin \mathrm{Err}_{\mathcal{M}}$.
   - If $I \in \mathcal{I}_{\mathrm{zt}}^c$ then w' is trivially accepted by $\mathrm{Ok}_{\mathcal{M}}^{\mathrm{R1}}$ and $\mathrm{Ok}_{\mathcal{M}}^{\mathrm{R2}}$ thus $w' \notin \mathrm{Err}_{\mathcal{M}}$.
2. If $\rho(w) \notin \mathrm{Runs}_{\mathcal{M}}$, there is some prefix $w' = w'' \cdot (I, m, t)$ of w such that $\rho(w') \notin \mathrm{Runs}_{\mathcal{M}}$ and $\rho(w'') \in \mathrm{Runs}_{\mathcal{M}}$. This is because at least all prefixes of the form $(\square, m, t)^*$ are encoding the empty run, which belongs to $\mathrm{Runs}_{\mathcal{M}}$.
   We look at the different cases for I to show that $w' \notin \mathrm{Ok}_{\mathcal{M}}$. Again, we let $f = \mathrm{frac}(t) \in \mathbb{F}$.
   - Suppose $I \in \mathcal{I}_{\mathrm{inc}}^c$. We have $\rho(w') \notin \mathrm{Runs}_{\mathcal{M}}$, $\rho(w'') \in \mathrm{Runs}_{\mathcal{M}}$, and $\rho(w') = \rho(w'') \xrightarrow{I} (s, \mathrm{val}(w'))$. This means that $\mathrm{val}(w')$ is not the correct valuation, that is $\mathrm{val}(w') \neq \mathrm{val}(w'')[c{+}{+}]$. From this, we deduce that f is active for c in w'', otherwise $\mathrm{val}(w')$ would be correct. Therefore $w' \in \mathrm{Err}_{\mathcal{M}}^{\mathrm{R1}}$, which implies $w' \notin \mathrm{Ok}_{\mathcal{M}}$.
   - Suppose $I \in \mathcal{I}_{\mathrm{dec}}^c$. There are two possible reasons for $\rho(w')$ not to be in $\mathrm{Runs}_{\mathcal{M}}$: either $\mathrm{val}(w')$ is wrong, or $\mathrm{val}(w'')(c) = 0$. In the first case, we deduce similarly to the increment case that f is inactive for c in w''. In the second case, it means that there are no active fractional part for c in w'', therefore f can only be inactive also. In both cases, f being inactive does not mean that $w' \in Err_2$, as f might be missing from the pool. However, this still means that $w' \notin \mathrm{Ok}_{\mathcal{M}}^{\mathrm{R2}}$ because $\mathrm{Ok}_{\mathcal{M}}^{\mathrm{R2}}$ only accepts if the last fractional part is active. Then we have that $w' \notin \mathrm{Ok}_{\mathcal{M}}$.
   - If $I \in \mathcal{I}_{\mathrm{zt}}^c$ then we immediately get a contradiction because by definition $\mathrm{val}(w')$ will be correct so $\rho(w')$ must be in $\mathrm{Runs}_{\mathcal{M}}$. ◀

Note that we cannot state $w' \in \mathrm{Ok}_{\mathcal{M}}$ for the first part and $w' \in \mathrm{Err}_{\mathcal{M}}$ for the second part, again because the pool may be missing the crucial fractional part needed for this. However, this lemma is enough to prove correctness of the reduction:

▶ **Theorem 15.** *$\mathcal{M}$ is bounded if and only if* Monitor *has a winning controller.*

Proof.

$\Rightarrow$ Assume $\mathcal{M}$ is bounded by k. We build a controller $\mathcal{A}$ for Monitor that will correctly detect the first time a mistake happens in the encoding of the run. This assumes that before that point the number of active fractional parts for a given counter never goes above k. This controller is a 1-resetting DTA using $4k$ clocks over input timed words with alphabet T and outputs in M.

Let $\mathcal{X} = \{x_i^c \mid 1 \leq i \leq k, c \in \mathcal{C}\}$ be the set of clocks of $\mathcal{A}$. Intuitively, clocks $x_1^c, \ldots, x_k^c$ are assigned to counter c and will "store" fractional parts that are active for c. By "storing" a fractional part f, we mean that this clock has value 0 exactly at timestamps whose fractional part is f. The state space of $\mathcal{A}$ keeps track of which clocks are considered active for their respective counters. The initial state is the empty set. $\mathcal{A}$ also ignores the initial pool of data and always outputs $\checkmark$ on those.

When reading an increment for $c \in \mathcal{C}$ with time t, $\mathcal{A}$ outputs $\lightning$ if any of the clocks indicated as active for c by the state is at value exactly 0, and $\checkmark$ otherwise. If it has output $\lightning$, it goes to a sink state that always output $\checkmark$. Otherwise, it resets the first clock x_i^c marked as inactive, and then it marks this clock as active. If there is no such free clock, it goes to an error state that outputs whatever.

Similarly, when reading a decrement for c with time t, $\mathcal{A}$ outputs $\lightning$ if no active clock has value 0, $\checkmark$ otherwise. By construction, no two clocks can have value 0 at the same time. If the output was $\lightning$, it goes to the same sink state as before. Otherwise, it removes that clock from the set of active clocks in the state.

Reading a zero-test transition for counter c makes $\mathcal{A}$ output $\checkmark$ and reset the set of active clocks for c to $\emptyset$.

First we show that $\mathcal{A}$ is *accurate* for runs bounded by k:

▶ **Lemma 16.** *For any play w that conforms to $\mathcal{A}$ that never has more than k active fractional parts for any counter, $\rho(w) \in \mathrm{Runs}_{\mathcal{M}}$ if and only if $\mathrm{proj}_{\mathrm{M}}(w) \in \checkmark^*$. Moreover, if $\mathrm{proj}_{\mathrm{M}}(w) \in \checkmark^*$, the valuation of a counter is exactly the number of clocks marked as active by the state of $\mathcal{A}$, each of those clocks stores one of the active fractional parts, and those clocks are pairwise different.*

Proof. We show this by induction on w. Again, this is trivial for any play of the form $(\square, \checkmark, t)^*$. Assuming the above holds for w and that $\mathcal{A}$ has always output $\checkmark$ so far, let us consider some continuation $w' = w \cdot (I, t)$ with $f = \mathrm{frac}(t)$.

Case 1a: If I is a c incrementing instruction and f is inactive for c in w, then $\mathrm{val}(w') = \mathrm{val}(w)[c{+}{+}]$ and therefore $\rho(w') \in \mathrm{Runs}_{\mathcal{M}}$. If $\mathrm{val}(w')(c) > k$ then the proof is finished and $\mathcal{A}$'s behaviour after this point does not matter. Otherwise, $\mathrm{val}(w)(c) < k$ and by induction hypothesis there are exactly $\mathrm{val}(w)(c)$ distinct fractional parts active for c in w and in exactly as many clocks of $\mathcal{A}$ assigned to c. Moreover, f is in none of them. In that case, $\mathcal{A}$ outputs $\checkmark$, and stores f in one free clock assigned to c, which we know there is at least one. Then all properties are satisfied by w'.

Case 1b: If I is a c incrementing instruction and f is active for c in w, then $\rho(w') \notin \mathrm{Runs}_{\mathcal{M}}$. Again by induction hypothesis f must be in one of the clocks of $\mathcal{A}$ assigned to c and marked as active. Therefore, $\mathcal{A}$ outputs $\lightning$ and we are done.

Cases 2a and 2b for a decrementing instruction with an active or inactive fractional part respectively are very similar.

Case 3 for a zero-test instruction is also easy: as zero-test instructions are always available under free-test semantics, $w' \in \text{Runs}_{\mathcal{M}}$ is immediate, $\mathcal{A}$ always outputs $\checkmark$, and all clocks are marked as inactive which coincide with the valuation being set to 0. ◀

Returning to the proof of Theorem 15, we now show that $\mathcal{A}$ is a winning controller for Monitor. Let w be a play.

If $\rho(w) \in \text{Runs}_{\mathcal{M}}$, then we know that there is at most k active fractional parts for any counter in all prefixes of w by boundedness assumption, and therefore that $\text{proj}_{\text{M}}(w) \in \checkmark^*$ by Lemma 16. Moreover, by Lemma 14, all prefixes of w are not in $\text{Err}_{\mathcal{M}}$. Clearly, w is not in $W_{\mathcal{M}}$.

If $\rho(w) \notin \text{Runs}_{\mathcal{M}}$, then by Lemma 14 there exists some prefix $w' = w'' \cdot (I, m, t)$ such that $\rho(w'') \in \text{Runs}_{\mathcal{M}}$, $\rho(w') \notin \text{Runs}_{\mathcal{M}}$, and $w' \notin \text{Ok}_{\mathcal{M}}$. Therefore, by Lemma 16, $\text{proj}_{\text{M}}(w'') \in \checkmark^*$ and $\mathcal{A}$ outputs $m = \chi$ on w'. w'' is not in $W_{\mathcal{M}}$ for the same reason as before, and since $w' \notin \text{Ok}_{\mathcal{M}}$, $w' \notin W_{\mathcal{M}}$ either. Then any continuation, including w, has proj_{M} of the form $\checkmark^* \cdot \chi \cdot \checkmark^+$ and therefore is not in $W_{\mathcal{M}}$.

$\Leftarrow$ Let ρ be an unbounded run of $\mathcal{M}$. Assume towards a contradiction that there exists some controller $\mathcal{A}$ that is winning for Monitor. Let $G \subset \text{Guard}(\mathcal{X})$ be the finite set of guards appearing in $\mathcal{A}$, and $k = |G|$. Given some guard $g \in G$ and a clock valuation ν, let $T(\nu, g) = \{t \in \mathbb{R}_{\geq 0} \mid \nu + t \models g\}$. As g is a conjunction of constraints, $T(\nu, g)$ is either empty, a singleton, or an interval. Thus, if there are two distinct $t, t' \in T(\nu, g)$ then any $t < t'' < t'$ is also in $T(\nu, g)$.

At some point, ρ goes from $k+1$ to $k+2$ for the valuation of some counter c using some incrementing instruction I. Let w be a timed word that correctly encodes ρ until just before I, with fractional parts $f_1, \ldots, f_{k+1}$ active at the end of w. For simplicity, we take $f_1 < \cdots < f_{k+1}$, and assume that w has those fractional parts in the pool. Moreover, we also put in the pool an inactive fractional part between every f_i and f_{i+1}.

Let us now consider what $\mathcal{A}$ does on this word. Necessarily $\mathcal{A}$ must have output only $\checkmark$ so far, otherwise Timer would win. On a new action (I, t), it must necessarily output χ if $\text{frac}(t) \in \{f_1, \ldots, f_{k+1}\}$ as those are active fractional parts, and therefore should not be used for a new increment. But on any other timestamp, if the corresponding fractional part is in the pool, it must output $\checkmark$ to not lose immediately. From $\mathcal{A}$'s current state, there are less than k transitions outputting χ. So at least one of them can be fired with at least two distinct active fractional parts. But as we have seen earlier, this means any timestamp between those two can also fire this transition. We have at least one inactive fractional part in the pool between the two active ones, and there is a timestamp with this fractional part that can fire the transition, outputting χ. Thus, we get a contradiction. ◀

Note that the existence of a controller and a strategy are not equivalent here: Monitor always has a winning strategy, but only has a winning controller if $\mathcal{M}$ is bounded.

7 Conclusion

The main contribution of this paper is to solve the problem left open in [13, 32] by proving that all the variants of timed synthesis problems are undecidable for winning sets specified already by NTA_1. The only exception is Theorem 11 in Section 6 where we need to extend (reachability) NTA_1 by a very limited form of ε-transitions. While we believe that these ε-transitions may be eliminated, we consider the current result as a satisfactory solution of

the open problem: all our undecidability results translate from timed setting to data setting where winning sets are specified using nondeterministic one-register automata (NRA_1), and neither ε-transitions nor guessing are needed there.

One of the motivations for studying timed/data synthesis in [13, 32] was a potential application to solving the *deterministic separability* question: given two nondeterministic NTA (resp. NRA) over finite timed (resp. data) words, with disjoint languages L_1, L_2, is there a DTA (DRA) whose language *separates* L_1 from L_2, namely includes one of them and is disjoint from the other. Decidability of resource-bounded timed/data synthesis is used in [13, 32] to obtain decidability of resource-bounded deterministic separability, where one a priori bounds the number of clocks/registers in a separating automaton. Decidability status of the unrestricted deterministic separability still remains open. Also a related problem of deterministic membership, where given a nondeterministic NTA, one asks if its language is accepted by some deterministic DTA, is undecidable even for NTA_1 [12]. On the other hand it decomes decidable if the number of clocks in a deterministic timed automaton is a priori bounded [12].

Knowing that NTA_1 winning sets yield ubiquitous undecidability of synthesis problems, a natural follow-up questions is to ask if the situation changes when one restricts to subclasses of NTA_1, for instance to *reachability* NTA_1 winning sets. Since deterministic separability over finite words reduces to timed games with reachability NTA winning conditions, this could help solve deterministic separability of NTA_1 languages.

Finally, we recall that we do not know if the games studied in this paper are determined, namely if one of the players has always a winning strategy.

References

1 Parosh Aziz Abdulla, Pavel Krcal, and Wang Yi. Sampled universality of timed automata. In *Foundations of Software Science and Computational Structures*, pages 2–16, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

3 Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Proc. of HSCC'99*, HSCC '99, pages 19–30, London, UK, UK, 1999. Springer-Verlag. doi:10.1007/3-540-48983-5_6.

4 Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. of SSSC'98*, volume 31 of *5th IFAC Conference on System Structure and Control*, pages 447–452, 1998. doi:10.1016/S1474-6670(17)42032-5.

5 Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems*, QEST '06, pages 125–126, Washington, DC, USA, 2006. IEEE Computer Society. doi:10.1109/QEST.2006.59.

6 Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundam. Inf.*, 36(2–3):145–182, November 1998. doi:10.3233/FI-1998-36233.

7 Sougata Bose, Thomas A. Henzinger, Karoliina Lehtinen, Sven Schewe, and Patrick Totzke. History-deterministic timed automata. *Log. Methods Comput. Sci.*, 20(4), 2024. doi:10.46298/LMCS-20(4:1)2024.

8 Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. Fault diagnosis using timed automata. In *Proc. of FOSSACS'05*, FOSSACS'05, pages 219–233, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/978-3-540-31982-5_14.

9 Thomas Brihaye, Thomas A. Henzinger, Vinayak S. Prabhu, and Jean-François Raskin. Minimum-time reachability in timed games. In Lars Arge, Christian Cachin, Tomasz Jurdziński, and Andrzej Tarlecki, editors, *Proc. of ICALP'07*, pages 825–837, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-73420-8_71.

10 J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. URL: http://www.jstor.org/stable/1994916.

11 Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *Proc. of CONCUR'05*, pages 66–80, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

12 Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Determinisability of One-Clock Timed Automata. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CONCUR.2020.42.

13 Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Timed games and deterministic separability. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 121:1–121:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.121.

14 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In Roberto Amadio and Denis Lugiez, editors, *Proc. of CONCUR'03*, pages 144–158, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

15 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.

16 Deepak D'souza and P. Madhusudan. Timed control synthesis for external specifications. In Helmut Alt and Afonso Ferreira, editors, *Proc. of STACS'02*, pages 571–582, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45841-7_47.

17 Léo Exibard. *Automatic Synthesis of Systems with Data. (Synthèse Automatique de Systèmes avec Données)*. PhD thesis, Aix-Marseille University, France, 2021. URL: https://tel.archives-ouvertes.fr/tel-03409602.

18 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over linearly ordered data domains. *Formal Methods Syst. Des.*, 61(2):290–337, 2022. doi:10.1007/S10703-023-00435-W.

19 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of data word transducers. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 24:1–24:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.24.

20 Diego Figueira, Piotr Hofman, and Sławomir Lasota. Relating timed and register automata. *Math. Struct. Comput. Sci.*, 26(6):993–1021, 2016. doi:10.1017/S0960129514000322.

21 Mark Jenkins, Joël Ouaknine, Alexander Rabinovich, and James Worrell. The church synthesis problem with metric. In *Computer Science Logic (CSL'11)-25th International Workshop/20th Annual Conference of the EACSL (2011)*, pages 307–321. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.CSL.2011.307.

22 Marcin Jurdziński and Ashutosh Trivedi. Reachability-time games on timed automata. In *Proc. of ICALP'07*, pages 838–849, Berlin, Heidelberg, 2007. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=2394539.2394637.

23 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

24 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. of CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011. doi:10.1007/978-3-642-22110-1_47.

25 Sławomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic*, 9(2, Article 10), 2008.

26 Sławomir Lasota, Mathieu Lehaut, Julie Parreaux, and Radosław Piórkowski. One-clock synthesis problems, 2026. arXiv:2601.04902.

27 Oded Maler and Amir Pnueli. On recognizable timed languages. In Igor Walukiewicz, editor, *Proc. of FOSSACS'04*, volume 2987 of *LNCS*, pages 348–362. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24727-2_25.

28 Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In Ernst W. Mayr and Claude Puech, editors, *Proc. of STACS'95*, pages 229–242, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

29 Richard Mayr. Undecidable problems in unreliable computations. *Theor. Comput. Sci.*, 297(1-3):337–354, 2003. doi:10.1016/S0304-3975(02)00646-1.

30 Brian Nielsen and Arne Skou. Automated test generation from timed automata. *International Journal on Software Tools for Technology Transfer*, 5(1):59–77, November 2003. doi:10.1007/s10009-002-0094-1.

31 Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 54–63. IEEE Computer Society, 2004. doi:10.1109/LICS.2004.1319600.

32 Radosław Piórkowski. *Simplification problems for infinite-state systems*. PhD thesis, University of Warsaw, 2022.

33 Alexander Rabinovich and Daniel Fattal. The Church synthesis problem over continuous time. *Log. Methods Comput. Sci.*, 21(3), 2025. doi:10.46298/LMCS-21(3:8)2025.

34 Philippe Schnoebelen. Lossy counter machines decidability cheat sheet. In Antonín Kučera and Igor Potapov, editors, *Reachability Problems*, pages 51–75, 2010. doi:10.1007/978-3-642-15349-5_4.

35 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006. doi:10.1007/11874683_3.

36 Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. Time to learn - learning timed automata from tests. In Étienne André and Mariëlle Stoelinga, editors, *Proc. of FORMATS'19*, pages 216–235, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-29662-9_13.

37 Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. An algorithm for learning real-time automata. In *Proc of. the Annual Belgian-Dutch Machine Learning Conference (Benelearn'078)*, 2007.

38 H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. In *Proc. of CDC'91*, volume 2 of *Proceedings of the 30th IEEE Conference on Decision and Control*, pages 1527–1528, December 1991. doi:10.1109/CDC.1991.261658.