HISTORY-DETERMINISTIC TIMED AUTOMATA [*]

SOUGATA BOSE ●[a], THOMAS A. HENZINGER ●[b], KAROLIINA LEHTINEN ●[c], SVEN SCHEWE ●[a], AND PATRICK TOTZKE ●[a]

[a] University of Liverpool
   e-mail address: {sougata,svens,totzke}@liverpool.ac.uk

[b] IST Austria
   e-mail address: tah@ist.ac.at

[c] CNRS, Aix-Marseille University, LIS
   e-mail address: lehtinen@lis-lab.fr

ABSTRACT. We explore the notion of history-determinism in the context of timed automata (TA) over infinite timed words. History-deterministic (HD) automata are those in which nondeterminism can be resolved on the fly, based on the run constructed thus far. History-determinism is a robust property that admits different game-based characterisations, and HD specifications allow for game-based verification without an expensive determinization step.

We show that the class of timed ω-languages recognized by HD timed automata strictly extends that of deterministic ones, and is strictly included in those recognized by fully non-deterministic TA.

For non-deterministic timed automata it is known that universality is already undecidable for safety/reachability TA. For history-deterministic TA with arbitrary parity acceptance, we show that timed universality, inclusion, and synthesis all remain decidable and are ExpTime-complete.

For the subclass of TA with safety or reachability acceptance, one can decide (in ExpTime) whether such an automaton is history-deterministic, and if so, it can be effectively determinized without introducing new automaton states.

1. INTRODUCTION

Automata offer paradigmatic formalisms both for specifying and for modelling discrete transition systems, *i.e.* for providing descriptive as well as executable definitions of formal languages. Given a finite or infinite word, an automaton specifies whether or not the word belongs to the defined language. Deterministic automata are executable, because the word can be processed left-to-right, with each transition of the automaton determined by the

current input letter. Descriptive automata allow the powerful concept of nondeterminism, which yields more succinct or even more expressive specifications.

The notion of *history-determinism* lies between determinism and nondeterminism. History-deterministic automata are still executable, provided the execution engine is permitted to keep a record of all past inputs. Formally, a strategy r (*a.k.a.* "resolver") is a function from finite prefix runs to transitions that suggests for each input word w a specific run $r^*(w)$ of the automaton over w, namely, the run that results from having the function r determine, after each input letter, the next transition based on the prefix of the word processed so far. An automaton is *history-deterministic* if there exists a resolver r so that for every input word w, the automaton has an accepting run over w iff the specific run $r^*(w)$ is accepting.

The concept of history-determinism was first identified in [HP06], where it was noted that for solving graph games, it is not necessary to determinize history-deterministic specifications of ω-regular winning conditions. For this reason, history-deterministic automata were called "good-for-games". The term "history-determinism" was first used by [Col09]. The concept itself has since been referred to as both "history-determinism" and "good-for-gameness." Since [BL21] recently showed that, in a general context of quantitative automata, the two notions do not always coincide (specifically: for certain quantitative winning conditions, history-determinism implies the "good-for-games" property of an automaton, but not vice versa), we follow their more nuanced terminology and use the term "history-determinism" to denote the existence of a resolver and "good-for-games" for automata that preserve the winner of games under composition, as required for solving games without determinization.

There is also a tight link between a variant of the Church synthesis problem, called *good-enough synthesis* [AK20], and deciding history-determinism. Church synthesis asks whether a system can guarantee that its interaction with an uncontrollable environment satisfies a specification language for all possible environment behaviours. This model assumes that the environment is hostile and will, if possible, sabotage the system's efforts. This pessimistic view can be counter-productive. In the canonical example of a coffee machine, if the users (the environment) do not fill in the water container, the machine will fail to produce coffee. Church synthesis would declare the problem unrealisable: the machine may not produce coffee for all environment behaviours. In the good-enough synthesis problem, on the other hand, such failures are acceptable, and we can still return an implementation that produces coffee (satisfies the specification) whenever the environment behaves in a way that allows the desired behaviour (fills in the water container). Deciding the good-enough synthesis problem for a deterministic automaton is polynomially equivalent to deciding whether a nondeterministic automaton of the same type is history-deterministic [FLW20, BL21, GJLZ21]. The decidability and complexity of checking history-determinism is therefore particularly interesting.

In this paper, we study, for the first time, history-determinism in the context of *timed* automata. In a timed word, letters alternate with time delays, which are nonnegative real numbers. The resolver gets to look not only at all past input letters, but also at all past time delays, to suggest the next transition. We consider timed automata over infinite timed words with standard ω-regular acceptance conditions [AD94]. For the results of this paper, it does not matter whether or not the sum of all time delays provided by an infinite input word is required to diverge.

Our results can be classified into two parts. The first part of our results applies to all timed automata, and sometimes more generally, to all labelled transition systems. In this part we are concerned with solving the quintessential verification problem for timed systems,

namely *timed language inclusion*, in the special case of history-deterministic (*i.e.* executable) specifications. Since universality is undecidable for general timed automata, so is the timed language-inclusion problem for non-deterministic specifications [AD94]. This is the reason why much previous work in timed verification has focused on identifying determinizable subclasses of timed automata, such as event-clock automata [AFH99], and on studying deterministic extensions of the timed-automaton model, such as deterministic two-way timed automata [AH92]. Determinizable specifications can be complemented, thus supporting the *complementation-based* approach to language inclusion: in order to check if every word accepted by the implementation A is also accepted by the specification B, first determinize and complement B, and then check the intersection with A for emptiness. We show that the history-determinism of specifications suffices for deciding timed language inclusion, which demonstrates that determinizability is not required. More precisely, we prove that if A is a timed automaton and B is a history-deterministic timed automaton, it can be decided in EXPTIME if every timed word accepted by A is also accepted by B (Corollary 5.5).

In contrast to the traditional complementation-based approach to language inclusion, the history-deterministic approach is *game-based*. Like the complementation-based approach, the game-based approach is best formulated in the generic setting of labelled transition systems with acceptance conditions, so-called *fair LTS*. The acceptance condition of a fair LTS declares a subset of the infinite runs of the LTS to be fair (a special case is *safety* acceptance, which declares all infinite runs to be fair). Given two fair LTS A and B, the language of A is included in the language of B if for every fair run of A there is a fair run of B over the same (infinite) word. A sufficient condition for the language inclusion between A and B is the existence of a fair simulation relation between the states of A and the states of B, or equivalently, the existence of a winning strategy for player p_B in the following 2-player *fair simulation game*: (i) every transition chosen by player p_A on the state-transition graph A can be matched by a transition chosen by player p_B on the state-transition graph B with the same label (letter or time delay), and (ii) if the infinite sequence of transitions chosen by p_A produce a fair run of A, then the matching transitions chosen by p_B produce a fair run of B [HKR97]. Solving the fair simulation game is often simpler than checking language inclusion; it may be polynomial where language inclusion is not (*e.g.* in the case of finite safety or Büchi automata), or decidable where language inclusion is not (*e.g.* in the case of timed safety or Büchi automata [TAKB96]).

We show that for all fair LTS A and all history-deterministic fair LTS B, the condition that the language of A is included in the language of B is equivalent to the condition that A is fairly simulated by B. This observation reduces the language inclusion problem for history-deterministic specifications to the problem of solving a fair simulation game between implementation and specification. The solution of fair simulation games depends on the complexity of the acceptance conditions of A and B, but is often simpler than the complementation of B, and fair simulation games can be solvable even in the case of specifications that cannot be complemented. In Section 4.2 we show the existence of such a timed language. The game-based approach to checking language inclusion, which requires history-determinism, is therefore more general, and often more efficient, than the traditional complementation-based approach to checking language inclusion, which usually requires full determinization. Indeed, history-determinism is exactly the condition that allows the game-based approach to language inclusion: for a given fair LTS B, if it is the case that B can fairly simulate all fair LTS A whose language is included in the language of B, then B must be history-deterministic (Theorem 3.4).

More generally, turn-based timed games for which the winning condition is defined by a history-deterministic timed automaton are no harder to solve than those with deterministic winning conditions: the winner of such a timed game can be determined on the product of the (timed) arena with the automaton specifying the winning condition. We conjecture that this is the case also for the concurrent timed games of [dAFH+03] (cf. Section 8). Timed games have also been defined for the synthesis of timed systems from timed I/O specifications. Again, we show that the synthesis game of [DM02] can be solved not only for I/O specifications that are given by deterministic timed automata, but more generally, for those given by history-deterministic timed automata (Theorem 7.2).

The second part of our results investigates the problem of deciding history-determinism for timed automata and the determinizability of history-deterministic timed automata. In this part, we have only partial results, namely results for timed safety and reachability automata. Timed safety automata, in particular, constitute an important class of specifications, as many interesting timed and untimed properties can be specified by timed safety automata if time is required to diverge [HNSY92, HKW95]. We prove that for timed safety automata and timed reachability automata, it can be decided in ExpTime if a given timed automaton is history-deterministic (Theorem 6.6). Checking history-determinism remains open for more general classes of timed automata, such as timed Büchi and coBüchi automata. We also show that every history-deterministic timed safety or reachability automaton can be determinized without increasing the number of automaton states, but with an exponential increase in the number of transitions or length of guards (Theorem 4.5). Since the question of determinizability is undecidable for non-deterministic timed reachability automata [Fin06], it follows from our result, that checking if a given non-deterministic timed reachability automaton has an equivalent HD timed reachability automaton is also undecidable. Finally, we show that if a timed safety or reachability automaton is good-for-games (in the sense explained earlier), then the automaton must be history-deterministic (Theorem 5.8). This implication is open for more general classes of timed automata.

All our results hold regardless of whether one assumes weak or strong progression of time (zero-delays are allowed, resp. forbidden) and also whether time must diverge, i.e., infinite words of finite duration (Zeno words) are considered or not.

Related Work. The notion of history-determinism was introduced independently, with slightly different definitions, by Henzinger and Piterman [HP06] for solving games without determinization, by Colcombet [Col09] for cost-functions, and by Kupferman, Safra, and Vardi [KSV06] for recognizing derived tree languages of word automata. Initially, history-determinism was mostly studied in the ω-regular setting, where these different definitions all coincide [BL19]. For some coBüchi-recognizable languages, history-deterministic automata can be exponentially more succinct than any equivalent deterministic automaton [KS15], and for Büchi and coBüchi automata, history-determinism is decidable in polynomial time [BK18, KS15]. For transition-based history-deterministic automata, minimisation is PTime [ARK19], while for state-based ones, it is NP-complete [Sch20]. Recently, the notion has been extended to richer automata models, such as pushdown automata [LZ22, GJLZ21] and quantitative automata [BL21, BL22], where deterministic and non-deterministic models have different expressivity, and therefore, allowing a little bit of non-determinism can, in addition to succinctness, also provide more expressivity.

Paper Structure. After defining preliminary notions we proceed to introduce history-determinism, and show a new, fair-simulation-based characterisation in Section 3. In Section 4 we consider the expressivity of history-deterministic(HD) timed automata(TA) with different Parity acceptance conditions. We show that history-deterministic TA with safety or reachability acceptance are determinizable. Section 5 considers questions concerning timed games, timed synthesis, and timed language inclusion and shows that history-determinism coincides with good-for-gameness for reachability and safety TA. In Section 6 we derive that one can decide, in EXPTIME, if a given safety or reachability TA is history-deterministic. In Section 7 we study synthesis games with winning conditions given by history-deterministic TA and show that solving the corresponding synthesis game remains in EXPTIME, as for deterministic TA. We conclude with a summary and some open problems and conjectures.

2. PRELIMINARIES

Numbers, Words. Let $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ denote the nonnegative integers and reals, respectively. For $c \in \mathbb{R}_{\geq 0}$ we write $\lfloor c \rfloor$ for its integer and $fract(c) \stackrel{\text{def}}{=} c - \lfloor c \rfloor$ for its fractional part.

An alphabet Σ is a nonempty set of letters. Σ_ε denotes $\Sigma \cup \{\varepsilon\}$. Σ^* and Σ^ω denote the sets of finite and infinite words over Σ, respectively and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ denotes their union. The empty word is denoted by ε, the length of a finite word v is denoted by $|v|$, and the n-th letter of a finite or infinite word is denoted by $w[n]$ (starting with $n = 0$).

Labelled Transition Systems, Languages, Fair Simulation. A *labelled transition system* (LTS) is a graph $S = (V, \Sigma, E)$ with set V of states and edges $E \subseteq V \times \Sigma \times V$, labelled by alphabet Σ. It is *deterministic* if for all $(s, a) \in V \times \Sigma$ there is at most one s' with $s \xrightarrow{a} s'$, and *complete* if for all $(s, a) \in V \times \Sigma$ there is at least one s' with $s \xrightarrow{a} s'$. We henceforth consider only complete LTSs. Together with an *acceptance condition* $Acc \subseteq E^\omega$ this can be used to define languages over Σ as usual: a word $w = l_0 l_1 \ldots \in \Sigma^\omega$ is accepted from s_0 if there is a path (also *run*) $\rho = s_0 \xrightarrow{l_1} s_1 \xrightarrow{l_2} s_2 \ldots$ that is accepting, i.e., in Acc. The *language* $L(s_0) \subseteq \Sigma^\omega$ of an initial state $s_0 \in V$ consists of all words for which there exists an accepting run from s_0. We will write $s \subseteq_L s'$ to denote language inclusion, meaning $L(s) \subseteq L(s')$. The acceptance condition Acc can be given by a parity condition but does not have to be. We consider in this paper especially reachability (does the run visit a state in a given target set $T \subseteq V$?) and safety conditions (does the run always stay in a "safe" region $F \subseteq V$?). An LTS together with an accepting condition is referred to as *fair* LTS [HKR97].

Fair simulations [HKR97] are characterised by simulation games on (a pair of) fair LTSs in which Player 1 stepwise produces a path from s, and Player 2 stepwise produces an equally labelled path from s'. Player 2 wins if she produces an accepting run whenever Player 1 does. That is, s is fairly simulated by s' (write $s \preceq s'$) iff Player 2 has a strategy in the simulation game so that, whenever the run produced by Player 1 is accepting then so is the run produced by Player 2 in response. Fair simulation $s \preceq s'$ implies language inclusion $L(s) \subseteq L(s')$ but not vice versa.

Timed Alphabets, Words, and LTSs. For any alphabet Σ let Σ_T denote the timed alphabet $\{(a,t)|a \in \Sigma, t \in \mathbb{R}_{\geq}0\}$. A timed word is a finite or infinite word $w \in (\Sigma_T)^{\infty}$ consisting of letters in Σ paired with distinct non-negative non-decreasing real-valued timestamps. We will also write $d_0a_0d_1a_1...$ to denote a timed word $(a_i, t_i) \in \Sigma_T^{\infty}$ where $t_0 = d_0$ and $t_{i+1} = t_i + d_{i+1}$. Conversely, the duration and the timed word of any sequence in $(\Sigma \cup \mathbb{R})^{\infty}$ is given inductively as follows. For any $d \in \mathbb{R}_{\geq 0}$, $\tau \in \Sigma$, $\alpha \in (\Sigma \cup \mathbb{R})^*$, and $\beta \in (\Sigma \cup \mathbb{R})^{\infty}$ let $duration(\tau) \stackrel{\text{def}}{=} 0$; $duration(d) \stackrel{\text{def}}{=} d$; $duration(\alpha\beta) = duration(\alpha) + duration(\beta)$; $tword(\varepsilon) = tword(d) \stackrel{\text{def}}{=} \varepsilon$; $tword(\alpha d) \stackrel{\text{def}}{=} tword(\alpha)$; and $tword(\alpha\tau) \stackrel{\text{def}}{=} tword(\alpha)(\tau, duration(\alpha))$. An infinite timed word of finite duration is called a *Zeno* word.

A *timed* LTS is one with edge labels in $\Sigma \uplus \mathbb{R}_{\geq 0}$, so that edges labelled by $\mathbb{R}_{\geq 0}$ (modelling the passing of time) satisfy the following conditions for all $\alpha, \beta, \gamma \in V$ and $d, d' \in \mathbb{R}_{\geq 0}$.

(1) (Zero-delay): $\alpha \xrightarrow{0} \alpha$,
(2) (Determinism): If $\alpha \xrightarrow{d} \beta \wedge \alpha \xrightarrow{d} \gamma$ then $\beta = \gamma$,
(3) (Additivity): $\alpha \xrightarrow{d} \beta \xrightarrow{d'} \gamma$ then $\alpha \xrightarrow{d+d'} \gamma$.

The timed language $L_T(s) \subseteq \Sigma_T^{\omega}$ of a state s consists of all the timed words read along accepting runs $L_T(s) \stackrel{\text{def}}{=} tword(L(s))$. We write $L(S)$ for the timed language of the initial state of the LTS S.

Timed Automata. Timed automata are finite-state automata equipped with finitely many real-valued variables called *clocks*, whose transitions are guarded by constraints on clocks. Constraints on clocks $C = \{x, y, \ldots\}$ are (in)equalities $x - y \triangleleft n$ where $x \in C$, $y \in C \cup \{0\}$, $n \in \mathbb{N}$ and $\triangleleft \in \{\leq, <\}$. Let $\mathcal{B}(C)$ denote the set of Boolean combinations of clock constraints, called *guards*. A clock *valuation* $\nu \in \mathbb{R}^C$ assigns a value $\nu(x)$ to each clock $x \in C$. We write $\nu \models g$ if ν satisfies the guard g. A timed automaton (TA) $\mathcal{T} = (Q, \iota, C, \Delta, \Sigma, Acc)$ is given by

- Q a finite set of states including an initial state ι;
- Σ an input alphabet;
- C a finite set of clocks;
- $\Delta \subseteq Q \times \mathcal{B}(C) \times \Sigma \times 2^C \times Q$ a set of transitions; each transition is associated with a guard, a letter, and a set of clocks to reset. A transition that reads letter $a \in \Sigma$ will be called an a-transition. We assume that for all $(s, \nu, a) \in Q \times \mathbb{R}_{\geq 0}^C \times \Sigma$ there is at least one transition $(s, g, a, r, s') \in \Delta$ so that ν satisfies g.
- $Acc \subseteq \Delta^{\omega}$ an acceptance condition.

Timed automata induce timed LTSs, and can thus be used to define timed languages, as follows. A *configuration* is a pair consisting of a control state and a clock valuation. These can evolve in two ways, as follows. For all configurations $(s, \nu) \in Q \times \mathbb{R}_{\geq 0}^C$,

- there is a *delay* step $(s, \nu) \xrightarrow{d} (s, \nu + d)$ for every $d \geq 0$, which increments all clocks by d.
- there is a *discrete* step $(s, \nu) \xrightarrow{\tau} (s', \nu')$ if $\tau = (s, g, a, r, s') \in \Delta$ is a transition so that ν satisfies g and $\nu' = \nu[r \to 0]$, that is, it maps r to 0 and agrees with ν on all other values.

Naturally, every delay d yields a unique successor configuration and so, for any two $d, d' \geq 0$ and valuations ν, ν', $\nu \xrightarrow{d} \xrightarrow{d'} \nu' \iff \nu \xrightarrow{d+d'} \nu'$. Hence, a TA indeed induces a timed LTS.

Discrete steps, however, are a source of nondeterminism: a configuration may have several a-successors induced by different transitions whose guards are satisfied. $\mathcal{T}$ is *deterministic* if

its induced LTS is deterministic, which is the case iff for every state s, all transitions from s have mutually exclusive guards.

A path $\rho = (s_0, \nu_0) \xrightarrow{l_1} (s_1, \nu_1) \xrightarrow{l_2} (s_2, \nu_2) \ldots$ is called *reduced* if it does not contain consecutive delay steps. It is a *run on* timed word $w \in (\Sigma_T)^\infty$ if $tword(l_1 l_2 \ldots) = w$. The acceptance condition is lifted to the LTS as expected. Namely, a run is *accepting* if $\rho \in Acc$. This way, the *language* $L_T(s, \nu) \subseteq \Sigma_T^\omega$ of a configuration (s, ν) consists of all timed words for which there exists an accepting run from (s, ν). The language of $\mathcal{T}$ is $L_T(\mathcal{T}) \overset{\text{def}}{=} L_T((\iota, 0))$, the languages of the initial configuration with state ι and all clocks set to zero.

Region Abstraction. The following is the standard definition of regions for timed automata (cf. [AD94], def. 4.3). Let $\mathcal{T} = (Q, \iota, C, \Delta, \Sigma, Acc)$ be a timed automaton and for any clock $x \in C$ let c_x denote the largest constant in any clock constraint involving x. Two valuations $\nu, \nu' \in \mathbb{R}_{\geq 0}^C$ are *(region) equivalent* (write $\nu \sim \nu'$) if all of the following hold.

(1) For all $x \in C$ either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or both $\nu(x)$ and $\nu'(x)$ are greater than c_x.
(2) For all $x, y \in C$ with $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $fract(\nu(x)) \leq fract(\nu(y))$ iff $fract(\nu'(x)) \leq fract(\nu'(y))$.
(3) For all $x \in C$ with $\nu(x) \leq c_x$, $fract(\nu(x)) = 0$ iff $fract(\nu'(x)) = 0$.

It follows that there are only finitely many equivalence classes w.r.t. $\sim$, called regions, for any given TA. Two configurations (s, ν) and (s', ν') are (region) equivalent, write $(s, \nu) \sim (s', \nu')$, if $s = s'$ and $\nu \sim \nu'$. Two runs are (region) equivalent if they have the same length and stepwise visit region equivalent configurations. Let $maxfrac(\nu) = \max\{fract(\nu(x)) \mid x \in C\}$ denote the maximal fractional value of any clock in configuration ν. We will make use of the following two properties.

Proposition 2.1.
(1) *For any valuation ν and $d \leq 1 - maxfrac(\nu)$ we have $\nu \sim \nu + d$.*
(2) *Suppose that $(p, \nu) \sim (p', \nu')$ and let $\rho \in (\Delta \cup \mathbb{R}_{\geq 0})^*$ satisfy $duration(\rho) < 1 - maxfrac(\nu)$, $duration(\rho) < 1 - maxfrac(\nu')$ and $(p, \nu) \xrightarrow{\rho} (q, \mu)$.*
     Then $(p', \nu') \xrightarrow{\rho} (q', \mu')$ for some $(q', \mu') \sim (q, \mu)$.

Proof sketch. Part 1 is immediate from the definition of regions.

Part 2 can be shown by induction on the length of ρ using the facts that region-equivalent configurations enable the same discrete transitions and that any delay decreases the duration of the remaining path by the same amount it increases clocks. $\qquad \square$

3. History-determinism

Informally, an automaton or LTS is history-deterministic if the nondeterminism can be resolved on-the-fly, based only on the history of the word and run so far. We give two equivalent definitions, each being more convenient than the other for some technical developments. The first one extends the original definition of history-deterministic automata (defined as good-for-games automata in [HP06]) to all fair LTSs.

Definition 3.1 (History-determinism)**.** A fair LTS $S = (V, \Sigma, E)$ is *history-deterministic* (from initial state $s_0 \in V$) if there is a *resolver* $r : E^* \times \Sigma \to E$ that maps every finite run and letter $a \in \Sigma$ to an a-labelled transition such that, for all words $w = a_0 a_1 \cdots \in L_T(s_0)$

the run ρ defined inductively for $i > 0$ by $\rho_{i+1} \stackrel{\text{def}}{=} \rho_i r(\rho_i, a_{i+1})$, is an accepting run on w from s_0.

Equivalently (from [BL19] for ω-regular automata), a resolver corresponds exactly to a winning strategy for Player 2 in the following *letter game*.

Definition 3.2 (Letter game)**.** The letter game on a fair LTS $S = (V, \Sigma, E)$ with initial state $s_0 \in V$ is played between Players 1 and 2. At turn i:

- Player 1 chooses a letter $a_i \in \Sigma$.
- Player 2 chooses an a_i labelled edge $\tau_i \in E$.

A play is a pair (w, ρ) where $w = a_0 a_1 \ldots$ is an infinite word and $\rho = \tau_0 \tau_1 \ldots$ is a run on w. A play is winning for Player 2 if either $w \notin L_T(s_0)$ or ρ is an accepting run on w from s_0.

In these and other games we consider, strategies for both players are defined as usual, associating finite histories (runs) to valid player choices. Now winning strategies for Player 2 in the letter game exactly correspond to resolvers for S and vice-versa.

Proposition 3.3. *Player 2 wins the letter game on a fair LTS S if and only if S is history-deterministic.*

While history-determinism is known to relate to fair simulation, in the sense that history-deterministic automata simulate deterministic ones for the same language [HP06], their relation has so far not been studied in more details. Below we show that history-determinism can equivalently be characterised in terms of fair simulation.

Theorem 3.4. *For every fair LTS S and initial state q the following are equivalent:*

(1) *S is history-deterministic.*
(2) *For all complete fair LTS S' with initial state q', $q' \subseteq_L q$ if and only if $q' \preceq q$.*

Proof (1) $\Longrightarrow$ (2). Fair simulation $q \preceq q'$ trivially implies $q \subseteq_L q'$ by definition.

For the other implication, assume that $q \subseteq_L q'$. By assumption (1) there exists a resolver, i.e. a winning strategy in the letter game. Player 2 can win the fair simulation game by ignoring her opponent's configuration and moving according to this resolver. By the completeness assumption on S', Player 1 can never propose a letter for which there is no successor in S'. So each player produces an infinite run on the same word w and the run produced by Player 2 is the same as that produced by the resolver in S'. If $w \in L_T(q)$ then it is in $L_T(q')$ and Player 2's run accepts. If $w \notin L_T(q)$ then Player 2 wins due to the fairness condition. In both cases she wins the fair simulation game and therefore $q \preceq q'$.

(2) $\Longrightarrow$ (1) If condition (2) holds for all complete fair LTSs then q can fairly simulate the one consisting of a single state with self-loops for all transitions of S whose acceptance condition contains exactly all accepting runs from q. Then the strategy for Player 2 in the fair simulation game can be used as a strategy in the letter game. $\qquad\square$

4. Expressivity

We consider the expressivity of history-deterministic TA in comparison to deterministic and fully non-deterministic variants, for different accepting conditions. In particular, we prove (in Section 4.1) that for safety and reachability acceptance, HD TA can be determinized, whereas for coBüchi and above they cannot (Section 4.2). Moreover, we show (in Section 4.3)
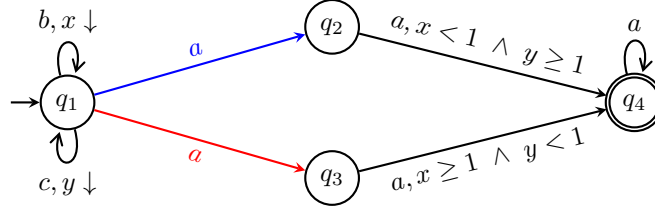
Figure 1: A history-deterministic timed reachability automaton where a resolver must distinguish between valuations where $x > y$ (go to q_3) and those where $x \le y$ (go to q_2).

that Parity HD TA are strictly less expressive than fully non-deterministic TA, even those with only a reachability acceptance.

4.1. Safety and Reachability HD TA are determinizable.

We start by showing that history-deterministic timed automata with safety acceptance are determinizable. To do so, we show (in Lemma 4.3) that these automata have simple resolvers, which only depend on the equivalence class of the current clock configuration with respect to the region abstraction. That is to say, the resolver only needs to know the integer part of clock values (up to the maximal value that appears in clock constraints) and the ordering of their fractional parts. We can then use such a simple resolver to determinize the automaton by adding guards that restrict transitions so that the automaton can only take one transition per region, as dictated by the resolver.

Before proving this formally, we first consider the reachability timed automaton depicted in Fig. 1. The accepted language consists of timed words that have a prefix of b and c events, followed by an infinite sequence of a events such that within one unit of time preceding the second a, there is either a b or a c event, but not both. The automata is history-deterministic: when reading the first a event, the resolver goes to q_2 if the previous letter was a b, and to state q_3 otherwise. Notice that this decision can be made only based on the current valuations and does not require the entire history, because the values of clocks satisfy $x \le y$ iff the last letter was b. The automaton can therefore be determinized by adding a guard $x - y \le 0$ to the blue edge (to q_2) and its negation to the red edge (to q_3).

Definition 4.1 (Run-trees). A *run-tree* on a timed word $u = (a_0, t_0)(a_1, t_1) \ldots$ from TA configuration (s_0, ν_0) is a tree where nodes are labelled by configurations, and edges by transitions such that

(1) The labels along every branch form a run on u from (s_0, ν_0).
(2) It is complete wrt. discrete steps: suppose the path leading towards some node is labelled by a run ρ which reads $tword(\rho) = (a_0, t_0) \ldots (a_i, t_i)$, ends in a configuration (s, ν), and has $duration(\rho) = t_{i+1}$. Then for every transition $\tau = (s, g, a_{i+1}, r, s') \in \Delta$ with $\nu \models g$ and so that $(s, \nu) \xrightarrow{\tau} (s', \nu')$, there is a τ-labelled edge to a new node labelled by (s', ν').

A run-tree is *reduced* if all its branches are. That is, there are no consecutive delay steps.

Notice that for every initial configuration and timed word, there is a unique reduced run-tree, all of whose branches are runs on the word (since we have no deadlocks), and vice versa, all reduced runs on the word appear as branches on the run-tree.

We extend the region equivalence from configurations to run-trees in the natural fashion: two run-trees are equivalent if they are isomorphic and all corresponding configurations are equivalent. That is, they can differ only in fractional clock values and the duration of delays.

The following is our key technical lemma.

Lemma 4.2. *Consider two region equivalent configurations $(s, \nu) \sim (s', \nu')$.*

For every timed word u there is a timed word u' so that the reduced run-tree on u from (s, ν) is equivalent to the reduced run-tree on u' from (s', ν').

Proof. It suffices to show that for some (not necessarily reduced) run-tree on u from (s, ν) there exists some equivalent run-tree from (s', ν') as this implies the claim by collapsing all consecutive delay steps and thus producing the reduced tree on both sides.

We proceed by stepwise uncovering the run-tree from (s, ν) for ever longer prefixes of u and constructing a corresponding equivalent run-tree from (s', ν'). The intermediate finite trees we build have the property that all branches have the same duration. In each round we extend all current leafs, in both trees, either by

(1) all possible non-deterministic successors (for the letter prescribed by the word u), in case the duration of the branch is already equal to the next time-stamp in u, or

(2) one successor configuration due to a delay, which must be *the same on all leafs*.

For the second case, the delays used to extend the two trees need not be the same because we only want to preserve region equivalence. Also, the delay chosen for the tree rooted in (s, ν) need not follow the timestamps in u but can be shorter, meaning the run-tree may not be reduced. The difficulty lies in systematically choosing the delays to ensure that the two trees remain equivalent, and secondly, that in the limit this procedure generates a run-tree on the whole word u from (s, ν). Together this implies the existence of a corresponding word u' and a run-tree from (s', ν').

Invariant. We propose a stronger invariant, namely that the relative orderings of the fractional values *in all leafs are the same on both sides*. To be precise, let's reinterpret a clock valuation as a function $\nu : C \times \mathbb{N} \to \{\bot\} \cup [0, 1)$, that assigns to every clock and possible integer value either a fractional value between 0 and 1, or $\bot$ (indicating that the given clock does not have the given integer value). This way for every clock x there is exactly one $n \in \mathbb{N}$ with $\nu(x, n) \neq \bot$ and the image $\nu(C \times \mathbb{N})$ has at most $|C| + 1$ different elements. For any ordered set $F = \{\bot < f_1 < f_2 < \cdots < f_\ell\} \supseteq \nu(C \times \mathbb{N})$ of fractional values, we can thus represent ν as a function $\hat{\nu} : C \times \mathbb{N} \to \{\bot, 1, \ldots, \ell\}$ that, instead of exact fractional clock values only yields their index in F (and maps $\bot \mapsto \bot$).

Consider some run-tree with leafs $(q_1, \nu_1)(q_2, \nu_2) \cdots (q_\ell, \nu_\ell)$ with combined fractional values $F = \bigcup_{j=1}^{\ell} \nu_j(C \times \mathbb{N})$, and an equivalent run-tree with leafs $(q_1', \nu_1')(q_2', \nu_2') \cdots (q_\ell' \nu_\ell')$ with combined fractional values $F' = \bigcup_{j=1}^{\ell} \nu_j'(C \times \mathbb{N})$. The two trees are *aligned* if for all $1 \leq j \leq \ell$, $\hat{\nu}_j = \hat{\nu}_j'$. Notice that this still allows the two trees to differ on their exact fractional values but now they must agree on the relative order of all contained clocks on leafs, and in particular which ones are maximal and therefore the closest to the next larger integer. We will always select a delay of $1 - \max\{F\}$ and $1 - \max\{F'\}$, respectively, in step 2 above.

To show the claim we produce the required run-trees starting in $(s, \nu) \sim (s', \nu')$. These are in particular two aligned run-trees on the empty word.

Assume two aligned trees as above, where leafs have fractional values $F = \{\bot < f_1 < f_2 < \cdots < f_m\}$ and $F' = \{\bot < f_1' < \cdots < f_m'\}$, respectively, and assume that the tree rooted in (s, ν) reads a strict prefix $(a_0, t_0), \ldots (a_i, t_i)$ of u.

Case 1: the duration of all branches in the first tree equals t_{i+1}, the timestamp of the next symbol in u. Then we extend each leaf in both trees by all possible a_{i+1}-successors. This will produce two aligned trees because each leaf configuration in one must be region equivalent to the corresponding configuration in the other, and therefore satisfies the same guards, enabling the same a_{i+1}-transitions leading to equivalent successors. Note also that all branches in each tree still have the same duration, as no delay step was taken.

Case 2: the duration of all branches in the first tree is strictly less than t_{i+1}. Then we extend all leafs in the tree from (s, ν) by a delay of duration $d = 1 - f_m$ and all leafs in the other tree by a delay of duration $d' = 1 - f'_m$. Naturally, this produces exactly one successor for each former leaf. The sets of new fractional values on leafs are $\bigcup_{j=1}^{\ell} (\nu_j + d)(C \times \mathbb{N}) = \{\bot < 0 < f_1 + d < \cdots < f_{m-1} + d\}$, and for any former leaf $(q, \mu) = (q_j, \nu_j)$ for $0 \le j \le \ell$, extended by a delay $(q, \mu) \xrightarrow{d} (q, \mu + d)$, we have

$$\hat{\mu}(x, n-1) = m \iff \widehat{(\mu + d)}(x, n) = 0 \tag{4.1}$$

and

$$\hat{\mu}(x, n) = j < m \iff \widehat{(\mu + d)}(x, n) = j + 1 \le m \tag{4.2}$$

Analogous equivalences hold for the corresponding step $(q, \mu') \xrightarrow{d'} (q, \mu' + d')$ on the other tree. Notice that the two cases above are exhaustive as again, for all $x \in C$ there is exactly one $n \in \mathbb{N}$ with $\mu(x, n) \ne \bot$. We aim to show that $\widehat{(\mu + d)} = \widehat{(\mu' + d')}$. Consider any $x \in C$ and $n \in \mathbb{N}$. We have that

$$
\begin{aligned}
\widehat{(\mu + d)}(x, n) = m \quad &\overset{(4.1)}{\iff} \quad \hat{\mu}(x, n+1) = 0 \\
&\overset{(IH)}{\iff} \quad \hat{\mu}'(x, n+1) = 0 \\
&\overset{(4.1)}{\iff} \quad \widehat{(\mu' + d')}(x, n) = m
\end{aligned}
$$

and

$$
\begin{aligned}
\widehat{(\mu + d)}(x, n) = j < m \quad &\overset{(4.2)}{\iff} \quad \hat{\mu}(x, n) = j + 1 \\
&\overset{(IH)}{\iff} \quad \hat{\mu}'(x, n) = j + 1 \\
&\overset{(4.2)}{\iff} \quad \widehat{(\mu' + d')}(x, n) = j < m
\end{aligned}
$$

It follows that $\widehat{(\mu + d)} = \widehat{(\mu' + d')}$ which means that the two trees are again aligned, as required.

To see why this procedure produces a run-tree on u (and an equivalent run-tree on some word u'), observe that there can be at most $|F| + 1$ many consecutive delay extensions according to step 2) before all integral clock values are strictly increased. $\qquad \square$

We are now ready to show that history-deterministic TA with safety acceptance have simple resolvers based on the region abstraction.

We call a resolver r *region-based* if it bases its decision only on the current letter and region. That is, if for any letter $a \in \Sigma$ and any two finite runs $(\iota, 0) \xrightarrow{\rho} (s, \nu)$ and $(\iota, 0) \xrightarrow{\rho'} (s', \nu')$ consistent with r and so that $(s, \nu) \sim (s', \nu')$, it holds that $r(\rho, a) = r(\rho', a)$.

Lemma 4.3. *Every history-deterministic TA with safety acceptance has a region-based resolver. The same is true for history-deterministic TA over finite words.*

Proof. Let r be a resolver for a history-deterministic safety TA $\mathcal{T}$.

We now build a resolver that only depends on the region of the current configuration. To do so, we choose a representative configuration within each region, which will determine the choice of the resolver for the whole region: For every region $R \in [Q \times \mathbb{R}_{\geq 0}^C]_\sim$, consider the configurations that are reached by at least one r-consistent run, and mark one of them m_R, if at least one exists, along with one r-consistent run ρ_R leading to the configuration m_R.

Let r' be the aspiring resolver that, when reading a letter a, considers the region R of the current configuration, and follows what r does when reading a after the marked r-consistent run ρ_R. We set $r'(\rho, a) \stackrel{\text{def}}{=} r(\rho_R, a)$ where R is the final region of the prefix-run ρ. Note that r' is well defined since it always follows transitions consistent with some r-consistent run and can therefore only visit marked regions.

We claim that r' is indeed a resolver. Towards a contradiction, assume that it is not a resolver, that is, there is some word $w \in L_T(\mathcal{T})$ for which r' builds a rejecting run. As $\mathcal{T}$ is a safety automaton, we can consider the last configuration (s, ν) along this run from which the remaining suffix au of w can be accepted [1].

Suppose that ρ is the prefix of the run built by r' on w, which ends in (s, ν) and let $\tau = r'(\rho, a)$ be the a-transition chosen by r'. We know that τ leads from (s, ν) to some configuration (s', ν') from where u is not accepted. By definition of r', there must be a marked configuration $m_R \sim (s, \nu)$ reached by some run ρ_R from which r chooses the same a-transition τ. By Lemma 4.2 there must be a word au' so that the run-tree on au from (s, ν) is equivalent to that on au' from m_R. This means that $au' \in L_T(m_R)$ and, as r is a resolver, there must be an accepting run that begins with a step $(m_R) \stackrel{\tau}{\longrightarrow} (m'_R)$. We derive that u also has an accepting run from (q, ν) that begins with τ, contradicting the assumption that (q, ν) is the last position on the run r' built on w so that its suffix can be accepted. Therefore, r' is indeed a resolver. $\qquad\square$

Using a technique of [BL22] we can show the existence of region-based resolvers also for HD timed reachability automata.

Lemma 4.4. *Every history-deterministic TA with reachability acceptance has a region-based resolver.*

Proof. Given a TA $\mathcal{T}$ with a reachability condition, we call a configuration (s, ν) *almost final* if it is universal ($L_T(s, \nu) = \Sigma_T^\omega$) and Player 2 wins the letter game on $\mathcal{T}$ starting at (s, ν). In particular, (s, ν) is almost final if s is a final state.

We first argue that this is a property of regions, rather than just configurations: if $(s, \nu) \sim (s, \nu')$ and (s, ν) is almost final then also (s, ν') must be almost final. Indeed, first observe that (s, ν') is universal by Lemma 4.2.

To see why there must also be a resolver from (s, ν'), we observe that the letter game starting from any universal configuration can be turned into an equivalent timed reachability game. This is because by the universality assumption, Player 1 can only win by keeping her opponent away from accepting configurations forever. The construction uses one extra clock that is used in particular to prevent Player 2 from making delay steps. For such games, both players have region-based winning strategies (see Theorem 5.3). Consequently, Player 2 can

---

[1]The fact that a rejecting run produced by a non-resolver must ultimately reach a configuration that cannot accept the remaining word also holds for TAs over finite words. However, this is *not* the case for infinite words defined via reachability acceptance.

use the same region-based strategy in the letter games from (s, ν') and (s, ν), which exists by our initial assumption.

We now argue that in the letter game for $\mathcal{T}$, any resolver must reach an almost final configuration as soon as Player 1 has played a word u for which some run ρ reaches an almost final configuration. Indeed, all continuations of u are in $L_T(\mathcal{T})$, so the configuration (s, ν) reached by the resolver must accept all continuations, and the resolver must represent a winning strategy in the letter game from (s, ν).

We now turn $\mathcal{T}$ into a new timed automaton $\mathcal{T}'$ over finite words with a single new accepting state that can be reached (via some new letter \$) exactly from all almost-final configurations. This is well defined because almost-final is a property of regions and therefore can be expressed by a clock constraint. Note that $\mathcal{T}'$ is history-deterministic because a resolver for $\mathcal{T}$ is also a resolver for $\mathcal{T}'$. This is because a resolver for $\mathcal{T}$ gives on a word u a run to an almost final configuration, whenever there exists one. Since $\mathcal{T}'$ is a history-deterministic TA on finite words, there exists a region-based strategy σ_1 in the letter game for $\mathcal{T}'$ by Lemma 4.3. We combine σ_1 with a region-based strategy σ_2 in the letter game from almost final configurations in $\mathcal{T}$. This yields a region-based strategy for Player 2 in the letter game for $\mathcal{T}$ which plays according to σ_1 while not almost final, and σ_2 from then onwards. This strategy must be a resolver: If Player 2 provides a word u such that, for no finite prefix there is a run ending in an almost final configuration, then u is not in $L_T(\mathcal{T})$ and Player 2 wins. If conversely, some prefix of u admits a run ending in an almost-final configuration, then σ_1 guarantees that Player 2 must be in such configuration after reading this prefix. From here σ_2 ensures that she wins whatever Player 1 plays. $\square$

We can now use the region-based solver to determinize history-deterministic safety and reachability TA.

Theorem 4.5. *Every history-deterministic safety and reachability TA is equivalent to a deterministic TA.*

Proof. Consider a history-deterministic TA $\mathcal{T} = (Q, \iota, C, \Delta, \Sigma, Acc)$, with a region-based resolver (as in Lemma 4.3) r, and let R be the region graph of $\mathcal{T}$. Define $\mathcal{T}' = (Q, \iota, C, \Delta', \Sigma, Acc)$ where $(q, g \wedge z, a, X, q') \in \Delta'$ for z a guard defining a region of R, that is, a guard that is satisfied exactly by valuations in R, if $(q, g, a, X, q') \in \Delta$ is the transition chosen by r in the region defined by the guard z. In other words, $\mathcal{T}'$ is $\mathcal{T}$ with duplicated transitions guarded so that a transition can only be taken from a region from which r chooses that transition. Observe that $\mathcal{T}'$ is deterministic: the guards describing regions are mutually exclusive, therefore the guards of any two transitions from the same state over the same letter have mutually exclusive guards.

As runs of $\mathcal{T}'$ corresponds to a run of $\mathcal{T}$ with added guards, $L_T(\mathcal{T}') \subseteq L_T(\mathcal{T})$. Conversely, if $w \in L_T(\mathcal{T})$, then its accepting run consistent with r is also an accepting run in $\mathcal{T}'$, since each transition along this run, being chosen by r, is taken at a configuration that satisfies the additional guards in $\mathcal{T}'$. We can therefore conclude that $L_T(\mathcal{T}) = L_T(\mathcal{T}')$. $\square$

While this determinization procedure preserves the state-space of the automaton, it multiplies the number of transitions (or the size of guards) by the size of the region abstraction. Thus, while history-deterministic reachability and safety TA are no more expressive than deterministic ones, they could still be more succinct, when counting transitions and guards. Notice that this construction requires guards of the form $x - y \triangleleft c$, where $y \in C$ is a clock (aka
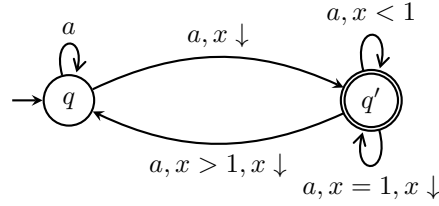
Figure 2: A history-deterministic timed co-Büchi automaton for L. The state q' has priority 0, i.e. is accepting, while the state q has priority 1.

diagonal constraints). These can be removed using the construction of [BPDG98] by adding additional states to obtain an equivalent deterministic TA without diagonal constraints.

4.2. **CoBüchi HD TA are not determinizable.** We show that history-deterministic timed automata are strictly more expressive than deterministic ones, for coBüchi acceptance and above. For simplicity let's first assume that our definition of timed words/languages excludes Zeno words. The case were these are allowed is a straightforward adjustment that we comment on at the end of the section. The rest of this subsection is a proof of the following theorem.

Theorem 4.6. *History-deterministic co-Büchi TA are more expressive than deterministic parity TA.*

Proof. The following timed language L over the singleton alphabet $\Sigma = \{a\}$ is recognized by a one-clock history-deterministic co-Büchi automaton yet not by any deterministic Parity timed automaton. In words, L asks to eventually see events a at unit distance. Formally,

$$ L \stackrel{\text{def}}{=} \{(a, t_0)(a, t_1)... \mid \exists i \in \mathbb{N}. \quad \forall n \in \mathbb{N}. \quad \exists j > i. \quad t_j - t_i = n\}. $$

We first show that L is recognized by the history-deterministic timed ω-automaton in Fig. 2. This automaton has an initial rejecting state q, from where there is a non-deterministic choice to either remain in this state or transition to an accepting state q', which resets the unique clock. There are two transitions to stay in the accepting state: one enabled when the clock value is smaller than 1, and one enabled at clock value 1, which also resets the clock. If the clock value grows larger than 1, the only enabled transition goes back to the initial state. Since this is a co-Büchi automaton, an accepting run must eventually remain in the accepting state.

First, this automaton recognizes L: if $w \in L$ then there is an accepting run that moves to state q' at time t, where it then remains since the clock x is reset at the occurrence of each event $(a, t + n)$ for $n \in \mathbb{N}$, so the clock value never grows larger than 1. Conversely, a word accepted by this automaton has a run that eventually moves to q' at a time t, and then remains in q'. For the run to stay in q', it must reset x at every time-unit after t, so $(a, t + n)$ must occur in the word for all $n \in \mathbb{N}$, that is, the word is in L.

We now argue that this automaton is also history-deterministic. Given a finite word read so far and a new letter a at time t_{new}, the resolver identifies the earliest time t_{early} such that a has so far occurred at time $t_{early} + n$ for all integers n such that $t_{early} + n \leq t_{new}$. Let r be the function that maps a run ρ ending in q to q' if $t_{new} = t_{early} + m$ for some integer m, and otherwise to the only other available transition.
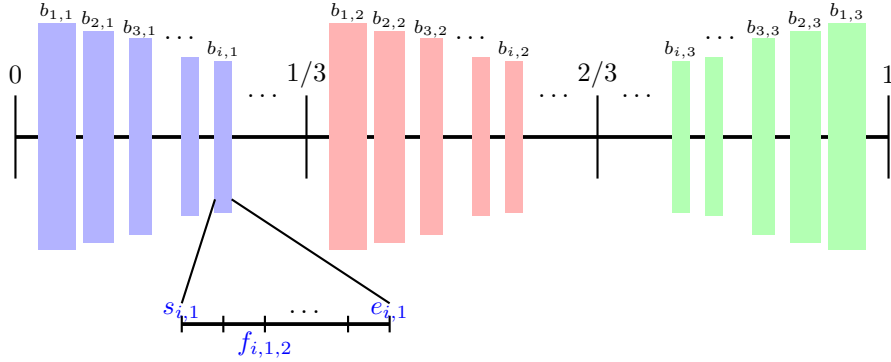
Figure 3: Blocks $b_{i,1}$, $b_{i,2}$, and $b_{i,3}$ within a unit time interval are displayed in blue, red and green respectively. Each block $b_{i,j}$ has equally spaced out events $f_{i,j,0} \ldots f_{i,j,r}$ where r is at most the number of regions.

We claim that this is indeed a resolver. If $w \in L$ then there is an earliest time t such that $(a, t + n)$ occurs in w for all integers n. Since t is minimal, eventually the resolver r will make its choice whether to move to q' over a letter (a, t_{new}) based on whether $t_{new} = t + m$ for some integer m. Since time progresses and $(a, t + n)$ occurs in w for all integers n, the run will eventually transition to q' at a time $t + m$ for some m. From there, since $(a, t + n)$ occurs in w for all integers n, the run over w remains in q' and is therefore accepting.

It remains to be shown that L is not recognized by a deterministic timed automaton. Suppose towards a contradiction that L is recognizable by some deterministic Timed Automaton D with Parity acceptance. Let r be the number of its regions.

We will construct two words w, w' such that $w \in L$ and $w' \notin L$, so that the run of D on w is region equivalent to the run on w'. The two words can only differ in the timing of events since there is only one letter in the alphabet.

Both words will be constructed on the fly, according to the following schema.

Consider the intervals and fractional values in Fig. 3; there are infinitely many disjoint intervals, $b_{i,j} = [s_{i,j}, e_{i,j}]$ so that all $b_{i,1}$ have start and endpoint strictly between 0 and $\frac{1}{3}$ and are increasing, i.e., $s_{i+1,1} > e_{i,1}$ for all i. Similarly, $b_{i,2} \subseteq [\frac{1}{3}, \frac{2}{3}]$, and $s_{i+1,2} > e_{i,2}$ for all i. The third sequence of intervals $b_{i,3} \subseteq [\frac{2}{3}, 1]$ have start and endpoint strictly between $\frac{2}{3}$ and 1 and are *decreasing*: $e_{i+1,3} < s_{i,3}$ for all i. Each interval $b_{i,j}$ contains equi-distant values $f_{i,j,0}, f_{i,j,1}, \ldots, f_{i,j,r}$ starting at $f_{i,j,0} = s_{i,j}$, where r is the number of regions.

We step-wise construct w (and w') together with the run of D on it. In every integral interval from $i - 1$ to i we place events as follows.

- start with a delay of $f_{i,1,1}$, followed by a discrete event a, then delay of $f_{i,1,2} - f_{i,1,1}$ followed by a, and so on. This induces a run of D on the prefix constructed and we continue constructing the prefix until the induced run closes a cycle in the region graph. Formally, there exists time $f_{i,1,k}$ and $f_{i,1,k+\ell}$ such that the run is in configuration $(q_{i,1,k}, \nu_{i,1,k})$ before reading the kth a and in configuration $(q_{i,1,k+\ell}, \nu_{i,1,k+\ell})$ at time $f_{i,1,k+\ell}$ such that $(q_{i,1,k}, \nu_{i,1,k}) \sim (q_{i,1,k+\ell}, \nu_{i,1,k+\ell})$ at time $f_{i,1,k+\ell}$. We denote by L_i the run between $f_{i,1,k}$ and $f_{i,1,k+\ell}$.
- Now we force the automaton to close the same cycle L_i, but with all events occurring at times in the interval $b_{i,2}$ (respectively $b_{1,2}$) in w (respectively w'). This can be done by

adding a time delay by $s_{i,2} - f_{i,1,k+\ell}$ in w followed by an event a at times $f_{i,2,\ell'}$ for all $\ell' \leq \ell$. We prove this formally in Lemma 4.7.

- Finally we force the automaton to close the same cycle L_i once more, with all times in interval $b_{i,3}$. This can be done by adding a time delay $s_{i,3} - f_{i,2,\ell}$ followed by events at times $f_{i,3,1}, f_{i,3,2}, \ldots f_{i,3,\ell}$. We prove the correctness of the construction in Lemma 4.7.

Consider the cycle L_i in the region graph obtained in step 1 above in the interval $[i-1, i]$, between $f_{i,1,k}$ and $f_{i,1,k+l}$. Note that the k and ℓ depends on i. However, we write k and ℓ without the dependency on i as we only reason about loops within an integral interval. The duration of the loop, denoted by $duration(L_i)$ is $f_{i,1,k+\ell} - f_{i,1,k}$. An important observation is that $duration(L_i) \leq e_{i,j} - s_{i,j}$ as the loop occurs within the interval $b_{i,1}$.

Lemma 4.7. *Let ν_i and ν_i' be the configurations reached by the run of D at times $i-1+f_{i,1,k}$ and $i-1+f_{i,1,k+\ell}$. Then $1 - maxfrac(\nu_i + d_{ij}) \geq duration(L_i)$, where $d_{ij} = s_{i,j} - f_{i,1,k}$ for $j \in \{2,3\}$.*

Furthermore, let ν_{ij} be the configuration reached by the run of D at time $i-1+f_{i,j,1}$, where $j = \{2,3\}$. The cycle L_i is executable from ν_{ij}.

Proof. We prove this lemma by induction on i. The case $i = 1$ is easy to see since $maxfrac(\nu_1 + d_{1j}) \leq s_{1,j}$ and therefore $1 - maxfrac(\nu_1 + d_{1j}) \geq 1 - s_{1,j} \geq e_{1,j} - s_{1,j} \geq duration(L_1)$.

Furthermore, $\nu_{12} = \nu_1' + d$, where $d = s_{1,2} - f_{1,1,k+\ell} \leq 1 - f_{1,1,k+\ell} \leq 1 - maxfrac(\nu_1')$. Therefore, by Proposition 2.1.1, $\nu_{12} \sim \nu_1' \sim \nu_1$. For $\nu = \nu_1$ and $\nu = \nu_{12}$, $1 - maxfrac(\nu) > e_{1,3} - s_{1,3} > duration(L_1)$ as $1 > e_{1,3}$ and $maxfrac(\nu) < s_{1,3}$. By applying Proposition 2.1.2, L_1 is executable from ν_{12} and ends in a configuration $\nu_{12}' \sim \nu_{12}$.

The configuration ν_{13} equals $\nu_{12}' + d'$, where $d' = s_{1,3} - f_{1,2,\ell} < 1 - maxfrac(\nu_{12}')$ as $maxfrac(\nu_{12}') \leq f_{1,2,\ell}$. Proposition 2.1.1 gives $\nu_{13} \sim \nu_{12}$, and $1 - maxfrac(\nu_{13}) \geq e_{1,3} - s_{1,3} \geq duration(L_1)$. By Proposition 2.1.2, we can conclude that L_1 is executable from ν_{13}.

To prove the inductive case, we bound the value of $maxfrac(\nu_i + d_{ij})$ for $j \in \{2,3\}$. Consider a clock $x \in C$ and the last time when it was reset. Either it was never reset or the reset occurred at time $f_{i',j',k'}$. For a clock that is never reset, the fractional part of its value at ν_i will be $f_{i,1,k}$. If the clock was last reset within some blue block, i.e, at time $i'-1+f_{i',1,k'}$, then either $i' < i$ (corresponds to previous blue blocks), or $k' < k$ (corresponds to previous ticks within the current blue block). In both cases, the $fract(x) = fract(f_{i,1,k} - (i'-1+f_{i',1,k'})) \leq f_{i,1,k}$.

Note that any reset to clock x in a previous red block must also be reset again in the corresponding green block as the runs in the red and green block are the same by construction. For a clock x last reset in some previous green block, i.e, at time $i'-1+f_{i',3,k'}$, $fract(x) = fract((i-1+f_{i,1,k}) - (i'-1+f_{i',3,k'})) = f_{i,1,k} + (1 - f_{i',3,k'})$. Furthermore, $f_{i',3,k'} > s_{i-1,3}$ as $i' \leq i$. Therefore, $fract(x) \leq 1 + f_{i,1,k} - s_{i-1,3} + 1$ which bounds $1 - fract(x) \geq s_{i-1,3} - f_{i,1,k}$. Combining all the possibilities for clock resets, we obtain $1 - maxfrac(\nu_i) \geq s_{i-1,3} - f_{i,1,k}$.

It is easy to see that for $j \in \{2,3\}$, $maxfrac(\nu_i + d_{ij}) \leq maxfrac(\nu_i) + d_{ij}$. Therefore, $1 - maxfrac(\nu_i + d_{ij}) \geq 1 - maxfrac(\nu_i) - d_{ij} \geq 1 - (f_{i,1,k} - s_{i-1,3} + 1) - (s_{i,j} - f_{i,1,k}) \geq s_{i-1,3} - s_{i,j} \geq e_{i,j} - s_{i,2}$. The last step follows from the fact that $e_{i,j} \leq s_{i-1,3}$ for $j \in \{2,3\}$. Note that the duration of the loop L_i is less than $e_{i,j} - s_{i,j}$ and thus completes the proof for first part of the lemma.

We now show that L_i is executable from ν_{i2} and ν_{i3}. First, $\nu_{i2} \sim \nu_i$ and $\nu_{i3} \sim \nu_{i2}$ by repeated application of Proposition 2.1.1. This is similar to the argument in the base
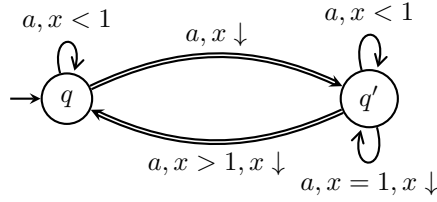
Figure 4: A history-deterministic timed co-Büchi automaton for $L \cup Z$ (the double edges are rejecting, all other accepting).

case. We just showed that $1 - maxfrac(\nu_{i2}) > s_{i-1,3} - s_{i,2} > e_{i,2} - s_{i,2} = duration(L_i)$. The same argument holds for ν_{i3} as well. Also, $maxfrac(\nu_i) \leq 1 - s_{i-1,3} + f_{i,1,k}$ and hence $1 - \max \nu_i \geq s_{i-1,3} - f_{i,1,k} < e_{i,3} - s_{i,3} < duration(L_i)$. Therefore, by Proposition 2.1.2, L_i is executable from both ν_{i2} and ν_{i3}.   □

Notice that the so-constructed word w is not in L because all $b_{i,j}$ are disjoint. The word w' will be constructed almost the same way, with the only exception that the first repetition of the cycle is moved not to $b_{i,2}$ but always the same interval, $b_{1,2}$. It is easy to see that Lemma 4.7 can be modified where $b_{i,2}$ is replaced everywhere by $b_{1,2}$. In particular this means that w contains an event at time $n + s_{1,2}$ for any $n \in \mathbb{N}$, and thus must be contained in L. Therefore, D has an accepting run on w' but the run on w' visits the same sequence of states as the run of D on w. Therefore, D must accept w as well, which is a contradiction proving that L is not accepted by any deterministic Timed Automaton with Parity acceptance.

This concludes the proof of Theorem 4.6.   □

Remark 4.8. In case our definition of timed words does allow for Zeno words, the argument above can be adjusted as follows. Instead of L, we consider the language $L \cup Z$, where Z denotes the set of all Zeno words. In fact the automaton depicted in Fig. 2 recognizes $L \cup Z$. However, the candidate resolver r proposed above may wait for ever in state q for the "oldest" fractional timestamp to re-appear while reading in a Zeno word. This results in a rejecting run on a word in the language, and so the r is no resolver. To fix this issue we adjust the automaton to the one depicted in Fig. 4. This is still a co-Büchi timed automaton, where the acceptance condition is via transitions: we want to finitely often use the edges that change state. Clearly this can be translated into a TA with state-based acceptance as before. The recognized language is still $L \cup Z$, the same as before. Note that runs that stay in q for ever are accepting Zeno words. For this automaton, the candidate resolver r (and its justification) is the same as above, except the run built for Zeno-continuations when in state q is actually accepting. The new automaton is therefore history-deterministic. A proof that $L \cup Z$ is not recognizable by any Parity DTA remains the same because the words constructed in the proof are non-Zeno.

4.3. HD TA are less expressive than fully non-deterministic TA.

Theorem 4.9. *History-deterministic reachability TA are less expressive than non-deterministic parity TA.*
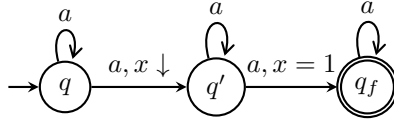
Figure 5: A non-deterministic timed reachability automaton for L'.

Proof. The following language L' over the singleton alphabet $\Sigma = \{a\}$ is recognized by a one-clock non-deterministic TA with reachability acceptance but not by any history-deterministic Parity TA. In words, L' asks to see two events a at unit distance. Formally,

$$L' \stackrel{\text{def}}{=} \{(\sigma_0, t_0)(\sigma_1, t_1)... \mid \exists i, j \in \mathbb{N}. \quad t_j - t_i = 1 \text{ and } \sigma_i = a \text{ and } \sigma_j = a\}.$$

The non-deterministic TA shown in Figure 5 accepts the language L' by guessing positions i by reading an a, resetting a clock x and checking that it sees an a at distance 1.

Assume towards a contradiction that there exists a HD TA H with k clocks and maximum constant in guards c_x, that recognizes L'. For all $i \leq k$ consider the finite word

$$w_i = \left(a, \frac{1}{k+1}\right) \cdots \left(a, \frac{k+1}{k+1}\right) \left(a, 1 + \frac{i}{k+1}\right)$$

that sees $k+1$ equi-distant events in the interval $[0,1]$ and then repeats the ith fractional value in the next integral interval. Note that $w_i \in L'$, for all $i \leq k$ and so the resolver gives a run on all such words. Note that the prefix up to time 1 is the same on all w_i and therefore the resolver gives the same run on the prefix until time 1. Consider the configuration ν reached by the resolver after reading the prefix up until and including the discrete event $(a, 1)$. Since H has k clocks and $k+1$ events a, there exists $j \leq k+1$ such that either no clock is reset at the jth a-event, or $j \leq k$ and the clocks reset at the jth a are reset again at a later a-event. This implies that the configuration ν reached at the end of the block is such that $\nu(x) \neq 1 - \frac{j}{k+1}$ holds for all clocks x. To see this note that if x was reset before the jth a, $\nu(x) \geq 1 - \frac{j-1}{k+1}$ and if x was reset after the jth a, $\nu(x) \leq 1 - \frac{j+1}{k+1}$. It follows that

$$\nu + \frac{j}{k+1} \sim \nu + \frac{j}{k+1} + \left(\frac{1}{2(k+1)}\right)$$

because for all clocks reset before the jth a, the value of x in both valuations is > 1 and for all clocks reset after the jth a, both valuations will satisfy $x < 1$.

Finally, let's take the word

$$w' = \left(a, \frac{1}{k+1}\right) \left(a, \frac{2}{k+1}\right) \cdots \left(a, \frac{k+1}{k+1}\right) \left(a, 1 + \frac{j}{k+1} + \frac{1}{2(k+1)}\right)$$

Clearly w' is not in L'. However, H must have a run on w' which follows the accepting run of H on w_j. The final step in this run can be executed because the two runs end up in equivalent configurations. A contradiction. □

By Theorems 4.6 and 4.9 we thus conclude that the classes of languages accepted by deterministic, history-deterministic and non-deterministic TAs are all different.

5. Timed Games and Composition

In this section we consider several games played on (LTSs of) timed automata and how they can be used to decide classical verification problems. We focus on turn-based games, although our techniques can be generalised to concurrent ones. We first look at language inclusion, then synthesis, and finally we consider good-for-games timed automata, that is, automata that preserve the winner when composed with a game and show that good-for-gameness and history-determinism coincide for both reachability and safety timed automata.

Definition 5.1 (Timed Games). A *timed game* is a turn-based two player game played on the configuration graph of a timed automaton. Formally, it consists of an arena $\mathcal{G} = (Q, \iota, C, \Delta, \Sigma, L)$, which is a TA except that the set Q of states is partitioned into those belonging to Player 1 (Q_1) and those belonging to Player 2 (Q_2), and that the acceptance condition L is a timed language (not some set of acceptable runs).

Configurations are defined as for TA. A timed game starts in the initial configuration $(\iota, \vec{0})$ and proceeds so that each round i from configuration c_i, the owner of the current state first advances time with a delay $d_i \in \mathbb{R}_{\geq 0}$ and follows it up with a discrete step from $c_i + d$ leading to a successor configuration c_{i+1} and emitting a letter $a_i \in \Sigma$. An infinite play is winning for Player 2 iff the word $d_0 a_0 d_1 a_1 \ldots$ produced is in L.

A *timed parity game* is a timed game where the acceptance condition L is given by a timed parity automaton. We assume w.l.o.g., that the configuration graph of the game and that of the parity automaton defining L is the same, so that a timed parity game is simply given as a timed game as above, and a colouring $priority : Q \mapsto \mathbb{N}$.

Remark 5.2. Notice that the definition of timed games above enforces that delays and discrete letters are emitted alternatingly. In particular, every play uniquely introduces an (infinite) timed word. The definition does not explicitly forbid these words to be Zeno (as for instance [CHP08] do) nor requires that only Player 1 advances time. To enforce that only Player 1 enforces delay, we can add a clock that is reset at every transition and is checked for 0 at every outgoing transition from a Player 2 vertex. To handle Zeno words, various winning conditions have been proposed that avoid that one player can always win by delaying. For example, as Zeno words are always winning for Player 2, or the player proposing 0 delay infinitely often loses. We refer to the introduction of [dAFH+03] for a summary of the different frameworks to handle Zeno words in the winning conditions.

We will occasionally use the fact that one can effectively determine the winner of a timed parity game.

Theorem 5.3 (Theorem 5, [dAFH+03]). *Consider a timed parity game with m states, n clocks, maximal constant k and c priorities.*
(1) *If Player i has a winning strategy then also one that is region-based, i.e., makes the same choices from all configurations in the same region.*
(2) *The game can be solved in time $\mathcal{O}((m \cdot n! \cdot 2n \cdot (2k + 1)n \cdot m \cdot c)^{(c+1)})$, therefore, in* ExpTime.

5.1. **Language Inclusion and Fair Simulation Games.** The connection between history-determinism and fair simulation, established in Theorem 3.4, allows to transfer decidability results to history-deterministic TA. Let's first recall that simulation checking is decidable for timed automata using a region construction [TAKB96]. This paper precedes the notion

of fair simulation (restricting Player 1 to fair runs) and is thus only applicable for safety conditions. However, the result holds for more general parity acceptance (for which each state is assigned an integer priority and where a run is accepted if the highest priority it sees infinitely often is even).

Theorem 5.4. *Checking fair simulation is in* ExpTime *for parity timed automata.*

Proof. For membership in ExpTime, it suffices to observe that the simulation game can be presented as timed parity game with suitable bounds, then apply Theorem 5.3.

Assume w.l.o.g. that both configurations to check are in the same timed automaton $\mathcal{A} = (Q, \iota, C, \Delta, \Sigma, priority)$ with $c = |priority(Q)|$ many colours.

The game is played on the product, with each player moving on their coordinate according to the TA semantics. The arena is bipartite and enforces alternation; Player 1 states are $Q_1 = (Q \times Q)$, Player 2 states are $(\Delta \times Q)$. The latter record Player 1's choice of discrete transition $t \in \Delta$. Player 1 first announces a delay (which affects all clocks and she remains in control) or the transition she wants. The latter choice resets some new clock and moves to a Player 2 owned state. Player 2 then has to move according to a transition carrying the same label as t, and updating both components along the two chosen transitions. This way, both players jointly construct on a timed word read in both copies.

The winning condition (parameter L for the timed parity game) encodes that if Player 1's run is accepting then so is that of Player 2. This can be achieved with a blow-up of the state space that is exponential only in c, the number of colours. Essentially, to implement the implication,

(1) increment all colours in Player 1's copy by one to negate the set of accepted runs;
(2) interpret the parity colouring(s) as Rabin chain condition with $c/2$ many indices[2].
(3) create a new Rabin acceptance condition for the product that implements the implication: for every index set (F_i, I_i) of Player 1, of finite and infinitely occurring states, resp., introduce a new Rabin pair $(F_i \times Q, I_i \times Q)$ and similarly, for every index set (F'_i, I'_i) of Player 2, introduce a new Rabin pair $(Q \times F'_i, Q \times I'_i)$. The size of the resulting index set is c.
(4) Finally, turn the whole game with Rabin acceptance back into one with parity acceptance using least appearance records [GH82, DJW97]. This results in a timed parity game with $m = |Q^2 + \Delta \times Q| \cdot c!$ states and $2c + 1$ colours.

The claim of ExpTime membership now follows by applying Theorem 5.3 as the obtained complexity can be bounded by $\mathcal{O}((m \cdot n! \cdot 2n \cdot (2k + 1)n \cdot 2c + 1)^{(2c+2)})$. The base in the above expression is dominated by $n! \cdot c!$ which can be bounded by $2^{n \cdot c}$ by using Stirling's approximation, which gives an exponential bound on the time complexity. $\quad\square$

Corollary 5.5. *Timed language inclusion is decidable and* ExpTime-*complete for history-deterministic TA. More precisely, given a TA S with initial state q and a history-deterministic TA S' with initial state q', checking if $q \subseteq_L q'$ holds is* ExpTime-*complete.*

Proof. As $\mathcal{B}$ is history-deterministic and by Theorem 3.4, we have $q \subseteq_L q'$ if, and only if, $q \preceq q'$. The result follows from Theorem 5.4. $\quad\square$

A matching lower bound holds even for safety or reachability acceptance, assuming that constants in transition guards are given in binary.

---

[2]More precisely, if c is even then both Rabin chain conditions have size $c/2$ and otherwise, if it is odd, they both have size $\lceil c/2 \rceil$.

Lemma 5.6. *Checking fair simulation between TA is* ExpTime-*hard already for reachability or safety acceptance, or over finite words.*

Proof. This can be shown by reduction from *countdown games* [JLS08], which are two-player games (Q, T, k) given by a finite set Q of control states, a finite set $T \subseteq (Q \times \mathbb{N}_{>0} \times Q)$ of transitions, labelled by positive integers, and a target number $k \in \mathbb{N}$. All numbers are given in binary encoding. The game is played in rounds, each of which starts in a pair (p, n) where $p \in Q$ and $n \leq k$, as follows. First Player 1 picks a number $l \leq k - n$, so that at least one $(p, l, p') \in T$ exists; then Player 2 picks one such transition and the next round starts in $(p', n + l)$. Player 1 wins iff she can reach a configuration (q, k) for some state q.

Determining the winner in a countdown game is ExpTime-complete [JLS08] and can easily be encoded as a simulation game between two TAs $\mathcal{A}$ and $\mathcal{B}$ as follows. Let $\mathcal{A}$ be the TA with no clocks and unrestricted (guards are *True*) self-loops for the two letters a and e; the idea is that Player 1 proposes l by waiting that long and then makes a discrete a-labelled move. Then Player 2, currently in some state p can update his configuration to mimic that of the countdown game, and punish (by going to a winning sink) if Player 1 cheated or the game should end. To implement this, $\mathcal{B}$ has two clocks: one to store n – the total time that passed – and one to store the current l, which is reset in each round.

Suppose Player 1 waits for l units of time and then proposes a. Player 2, currently in some state p will have

- a and e-labelled transitions to a winning state with a guard that verifies that there is no transition (p, l, p'). This can be done by checking the guard $\wedge_i (x_2 \neq l_i)$ for all l_i such that there exists p' with an edge (p, l_i, p') in the countdown game.
- a-labelled transitions to a state p', with a guard that verifies that some $(p, l, p') \in T$ exists, and which resets clock x_2.
- a, and e-labelled transitions to a winning state guarded by $x_1 > k$. This enables Player 2 to win if the global time has exceeded the target k.

The only way that Player 1 can win is by following a winning strategy in the countdown game and by playing the letter e once $\mathcal{B}$ is in a configuration (q, k). Player 2 will not be able to respond. □

5.2. Composition with Games.

Implicitly, at the heart of these reductions is the notion of composition: the composition of the game to solve with a history-deterministic automaton for the winning condition yields an equivalent game with a simpler winning condition. We say that an automaton is *good-for-games* if this composition operation preserves the winner of the game for all games. While history-determinism always implies good-for-gameness, the converse is not necessarily true. While the classes of history-deterministic and good-for-games automata coincide for ω-regular automata [BL19], this is not the case for quantitative automata [BL21], which can be good-for-games without being history-deterministic. We argue that for reachability and safety timed automata, good-for-gameness and history-determinism coincide.

Intuitively, the composition of a timed automaton $\mathcal{T}$ and a timed game $\mathcal{G}$ consists of a game in which the two players play on $\mathcal{G}$ while Player 2 must also build, letter by letter, a run of $\mathcal{T}$ on the outcome of the game in $\mathcal{G}$.

Definition 5.7 (Composition). Given a TA $\mathcal{T}$ and a timed game $\mathcal{G}$ with winning condition $L_T(\mathcal{T})$, the composition $\mathcal{T} \circ \mathcal{G}$ consists of a game played on the product of the configuration

spaces of $\mathcal{T}$ and $\mathcal{G}$, starting from the initial state of both, in which, at each turn i, from a configuration (c_i, c'_i), the owner of the current $\mathcal{G}$-state chooses a time delay $d_i \in \mathbb{R}_{\geq 0}$ advancing all clocks on both sides, and chooses a move in $\mathcal{G}$ to a successor-configuration c_{i+1}, producing a letter a_i, and then Player 2 chooses a transition over a_i enabled at the current $\mathcal{T}$-configuration c'_i, leading to a successor-configuration c'_{i+1}. The game then proceeds from (c_{i+1}, c'_{i+1}).

Player 2 wins infinite plays if the run built in $\mathcal{T}$ is accepting, and loses if it is rejecting or if she cannot move in the $\mathcal{G}$-component.

Observe that if Player 1 wins in $\mathcal{G}$, then he also wins in $\mathcal{T} \circ \mathcal{G}$ with a strategy that produces a word not in $L_T(\mathcal{T})$ in $\mathcal{G}$, as then Player 2 can not produce an accepting run in $\mathcal{T}$.

Boker and Lehtinen [BL21, Lemma 7] show that for (quantitative) automata for which the letter-game is determined, (threshold) history-determinism coincides with good-for-gameness. The lemma is stated for quantitative automata, where thresholds are relevant; in the Boolean setting, it simply states that the determinacy of the letter game implies the equivalence of history-determinism and good-for-gameness. In our timed setting, a similar argument, combined with the determinacy of the letter game for safety and reachability TA, gives us the following.

Theorem 5.8. *Let $\mathcal{T}$ be a safety or reachability TA. The following are equivalent:*

(1) *$\mathcal{T}$ is history-deterministic.*
(2) *For all timed games $\mathcal{G}$ with winning condition $L_T(\mathcal{T})$, whenever Player 2 wins $\mathcal{G}$, she also wins $\mathcal{T} \circ \mathcal{G}$.*

Proof. $(1) \implies (2)$ If $\mathcal{T}$ is history-deterministic, the resolver can be used as a strategy in the $\mathcal{T}$ component of $\mathcal{T} \circ \mathcal{G}$. When combined with a winning strategy in $\mathcal{G}$ that guarantees that the $\mathcal{G}$-component produces a word in $L_T(\mathcal{T})$, the resolver guarantees that the $\mathcal{T}$-component produces an accepting run, thus giving the victory to Player 2.

$(2) \implies (1)$ Towards a contradiction, assume $\mathcal{T}$ is not history-deterministic, that is, by determinacy of the letter game from Remark 6.2, that Player 1 has a winning strategy σ in the letter game. Now consider the game $\mathcal{G}_\sigma$, without clocks or guards, in which positions, all belonging to Player 1, consist of the prefixes of timed words played by σ, with moves $w \xrightarrow{(t,a)} w(t,a)$. As σ is winning for Player 1, all maximal paths in $\mathcal{G}_\sigma$ are labelled by a timed word in $L_T(\mathcal{T})$, so $\mathcal{G}_\sigma$ is winning for Player 2.

We now argue that Player 1 wins $\mathcal{T} \circ \mathcal{G}_\sigma$ by interpreting Player 2's moves in the $\mathcal{T}$ component as her moves in the letter game, and choosing moves in $\mathcal{G}$ mimicking the letter dictated by σ. Then, if Player 2 could win against this strategy in $\mathcal{T} \circ \mathcal{G}_\sigma$, she could also win against σ in the letter game by interpreting Player 1's choices of letters as moves in $\mathcal{G}$, and responding with the same transition as she plays in the $\mathcal{T}$ component of $\mathcal{T} \circ \mathcal{G}_\sigma$. Such a strategy is a valid strategy in the letter game on $\mathcal{T}$, and while it might not be winning in general, it is winning against σ, contradicting that σ is a winning strategy for Player 1. $\square$

This proof fails for acceptance conditions beyond safety and reachability, as it isn't clear whether timed Büchi and coBüchi automata define Borel sets. If this was the case then history-deterministic timed automata would be exactly those that preserve winners in composition with games, as is the case in the ω-regular setting.

6. Deciding History-determinism

Recall the letter game characterisation of history-determinism: Player 1 plays timed letters and Player 2 responds with transitions. Player 2 wins if either the word is not in the language of the automaton, or her run is accepting. As TA are not closed under complement, it isn't clear how to solve this game directly. Bagnol and Kuperberg [BK18] introduced *token games*, which are easier to solve, but which coincide with the letter game for various types of automata[BK18, BKLS20, BL22].

In this section we show that the simplest token game, called the one-token game, characterises history-determinism for safety and reachability LTSs, and therefore TAs. The case of reachability automata was already shown in [BL22], but for self-containment we include the proof here, generalised to fair LTSs. This strengthens and simplifies the result of the conference version [HLT22], which used the two-token game to characterise the history-determinism of reachability automata.

Definition 6.1 (1-token game [BK18]). Given a fair LTS $S = (V, \Sigma, E)$ with initial state $s_0 \in V$, the game $G_1(S)$ proceeds in rounds. At each round i:

- Player 1 plays a letter $a_i \in \Sigma$
- Player 2 plays a transition τ_i in S
- Player 1 plays a transition τ_i' in S

This way, Player 1 chooses an infinite word $w = a_0 a_1 \dots$ and a run $\rho' = \tau_0' \tau_1' \tau_2' \dots$, and Player 2 chooses a run $\rho = \tau_0 \tau_1 \dots$. The play is winning for Player 1 if ρ' is an accepting run over $t_0 a_0 \dots$ from s_0 but ρ is not. Else it is winning for Player 2.

Given a TA $\mathcal{T}$, we write $G_1(\mathcal{T})$ to mean the 1-token game on the fair LTS induced by $\mathcal{T}$.

Remark 6.2. $G_1(S)$ and the letter game are determined for any fair LTS S with any Borel-definable acceptance condition [Mar75]. In particular, the letter game is determined for both safety and reachability TA $\mathcal{T}$. Indeed, the winning condition for Player 2 is a disjunction of the complement of $L_T(\mathcal{T})$ and of the acceptance condition of $\mathcal{T}$. Then, as long as $L_T(\mathcal{T})$ is Borel, by the closure of Borel sets under complementation and disjunction, the letter-game is Borel, and therefore determined, following Martin's Theorem [Mar75]. If time is not required to diverge, then reachability timed languages and safety timed languages are clearly Borel. Since words in which time diverges are also Borel (they can be seen as the countable intersection of words where time reaches each unit time), this remains the case when we require divergence.

$G_1(S)$ was shown to characterise history-determinism for a number of quantitative automata in [BL22]. In Lemma 6.3 below we show, using similar proof techniques, that this is also the case for all safety LTSs. The key observation is that for Player 2 to win the letter game, it suffices that she avoids mistakes. We then show that a winning strategy for her in $G_1(S)$ can be used to build such a strategy.

Lemma 6.3. *Given a fair LTS S with a safety acceptance condition, or interpreted over finite words, Player 2 wins $G_1(S)$ if and only if S is history-deterministic.*

Proof. If S is history-deterministic then Player 2 wins $G_1(S)$ by using the resolver to choose her transitions. This guarantees that for all words in $L(S)$ played by Player 1, her run is accepting, which makes her victorious regardless of Player 1's run.

For the converse, if Player 2 wins $G_1(S)$ with strategy σ, consider the following family of *copycat strategies* for Player 1: at first, Player 1 plays σ and chooses the same transitions

as Player 2. If Player 2 ever chooses a transition τ from a configuration c that is not *language-maximal*, that is, moves to a configuration c' that does not accept some word w that is accepted by some other configuration c'' reachable by some other transition τ' from c, we call such a move *non-cautious*, and Player 1 stops copying Player 2 and instead chooses τ'. From there, Player 1 wins by playing w and an accepting run on w from c''. Since Player 2 wins $G_1(S)$, her winning strategy σ does not play any non-cautious moves against copycat strategies.

Then, she can use σ in the letter-game, by playing as σ would play in $G_1(S)$ if Player 1 copies her transitions. This guarantees that she never makes a non-cautious move, and, in particular, for S with a safety acceptance condition, never moves out of the safe region of the automaton unless the prefix played by Player 1 has no continuations in $L(S)$. This is a winning strategy in the letter-game, so S is history-deterministic. Similarly, for S interpreted over finite words, the letter game is a safety game, and a strategy that never makes a non-cautious move remains in the safe region of this game, and is therefore winning. $\square$

This argument does not work for reachability TA: it is no longer enough for Player 2 to avoid bad moves to win; she needs to also guarantee that she will actually reach a final state.

Given a fair LTS S with a reachability acceptance condition, we call a state q *almost final* if, from q, there is an accepting run on all words and Player 2 wins the letter game starting from S^q. In particular, every final state is also almost final.

Lemma 6.4. *Given an S with a reachability acceptance condition, let S' be S where every almost final state of S is made final in S. Then:*

(1) *If Player 2 wins $G_1(S)$ on infinite words, then Player 2 also wins $G_1(S')$ on finite words;*
(2) *If S' on finite words is history-deterministic, then so is S on infinite words.*

Proof. (1) Let σ be winning strategy for Player 2 in $G_1(S)$ on infinite words. Let the strategy σ' for Player 2 on $G_1(S')$ on finite words simply copy σ.

If Player 2's run in a play of $G_1(S')$ that agrees with σ' reaches a state that was almost final in S, and therefore final in S', it is winning for Player 2. If Player 2's run in a play of $G_1(S')$ that agrees with σ' does not reach such a state, we argue that Player 1's run also does not reach such a state. Indeed, towards a contradiction, assume that after some finite play π, Player 2's run has reached a state q_2 that is not final, while Player 1's run has reached a state q_1 that is final. Then, π is also a play prefix that agrees with σ in $G_1(S)$. We argue that Player 1 can then win in $G_1(S)$, a contradiction: since q_2 is not almost final, he can play letters so that Player 2 does not build an accepting run; meanwhile, from q_1, he can use the strategy witnessing that it is almost final to build an accepting run on any word.

(2) Let σ' be Player 2's winning strategy in the letter-game on S' on finite words. In the letter-game on S on infinite words, Player 2 follows σ' until the prefix of the word is in $L(S')$, at which point her run must have reached an almost final state of S, due to σ' being winning. From there, she can play the strategy witnessing that the state is almost final, and win. If Player 1 never plays a prefix in $L(S')$ then the word he plays is not in $L(S)$, and Player 2 wins. $\square$

Theorem 6.5. *Given a fair LTS S with a reachability acceptance condition, Player 2 wins $G_1(S)$ if and only if S is history-deterministic.*

Proof. One direction is immediate since if Player 2 wins in the letter game, she can use the same strategy to win in $G_1(S)$ by ignoring Player 1's run.

For the other direction, if Player 2 wins $G_1(S)$, she also wins $G_1(S')$ on finite words, for S' defined as in Lemma 6.4 above. Then, S' is history-deterministic from Lemma 6.3, and, again from Lemma 6.4, S is also history-deterministic. □

We now consider the problem of deciding whether a given safety or reachability TA is history-deterministic. We use the observation that the k-token games played on LTSs induced by TA can be expressed as a timed parity game from [CHP08] played on the $(k + 1)$-fold product.

Theorem 6.6. *Given a safety or reachability TA, deciding whether it is history-deterministic is decidable in* EXPTIME.

Proof. From Lemma 6.3 and Lemma 6.4, deciding the history-determinism of a safety or reachability TA $\mathcal{T}$ reduces to solving $G_1(\mathcal{T})$, which is a timed game played on the product of two copies of $\mathcal{T}$ (plus some intermediate states to encode the interaction in each round). The winning condition consists of a Boolean combination of safety or reachability conditions.

In the same way as done in the proof of Theorem 5.4, we can implement $G_1(\mathcal{T})$ as a timed parity game where the set states grows exponentially only in the number of colours in the parity condition. The resulting timed parity game can be solved in time exponential in the number of clocks c (see Theorem 5.3). □

As explained in the introduction, this also solves the good-enough synthesis problem of deterministic safety and reachability TA.

7. SYNTHESIS

We show that as is the case in the regular [HP06], pushdown [LZ22], cost function [Col09], and quantitative [BL21] settings, synthesis games with winning conditions given by history-deterministic TA are no harder to solve than those with winning condition given by deterministic TA.

Definition 7.1 (Timed synthesis game). Given a timed language $L \subseteq (\Sigma_I \times \Sigma_O)_T^\omega$, the synthesis game for L proceeds as follows. At turn i:
- Player 1 plays a delay d_i and a letter $a_i \in \Sigma_I$
- Player 2 plays a letter $b_i \in \Sigma_O$.

Player 2 wins if $d_0 \binom{a_0}{b_0} d_1 \binom{a_1}{b_1} \ldots \in L$ or if time does not progress. If Player 2 has a winning strategy in the synthesis game, we say that L is *realisable*.

Theorem 7.2. *Given a history-deterministic timed parity automaton $\mathcal{T}$, the synthesis game for $L_T(\mathcal{T})$ is decidable and* EXPTIME-*complete*.

The proof below follows a similar reduction to one in [LZ22], in which the nondeterminism of the automaton is moved into Player 2's output alphabet, forcing her to simultaneously build a word in the winning condition and an accepting run witnessing this. Since accepting runs are recognized by deterministic automata, this reduces the problem to the synthesis problem for deterministic timed automata. The lower bound follows from the EXPTIME-completeness of synthesis for deterministic TA [DM02].

The EXPTIME decidability of universality for history-deterministic TA follows both from the decidability of language inclusion in the previous section and from the decidability of synthesis: the universality of $\mathcal{T}$ reduces to deciding the winner of the synthesis game over $\{\binom{w}{w} \mid w \in L_T(\mathcal{T})\}$, recognized by a history-deterministic TA if $\mathcal{T}$ is history-deterministic.

Proof of Theorem 7.2. For the upper bound, we reduce the problem to solving synthesis games for deterministic timed parity automata, which is in ExpTime [DM02].

Let $\mathcal{T} = (S, \iota, C, \Delta, \Sigma, Acc)$ be a timed automaton. Let $\mathcal{T}'$ be the deterministic timed automaton $(S, \iota, C, \Delta', \Sigma \times \Delta, Acc)$ where:

$$\Delta' = \{(s, g, (\sigma, (s, g, \sigma, c, s')), c, s') | (s, g, \sigma, c, s') \in \Delta\}$$

In other words, $\mathcal{T}'$ is a deterministic automaton with the state space of $\mathcal{T}$, over the alphabet $\Sigma \times \Delta$, where the transition in the input letter dictates the transition in the automaton. The language of $\mathcal{T}'$ is the set of words (w, ρ) such that there is an accepting run of $\mathcal{T}$ over w along the transitions of ρ.

We now claim that given a history-deterministic automaton $\mathcal{T}$ with resolver r, Player 2 wins the synthesis game on $\mathcal{T}$ if and only if she wins it on $\mathcal{T}'$. First assume that Player 2 wins the synthesis game for $\mathcal{T}$ with a strategy s. Then, to win the synthesis game for $\mathcal{T}'$, at each turn i, after Player 1 plays d_i and a_i, she needs to make two choices: she must choose both a response letter b_i and a transition in $\mathcal{T}$ over (a_i, b_i). Given Player 1's move and the (first component of the) word built so far, she can use the strategy s to choose the response letter b_i; this guarantees that the first component of the play is a word accepted by $\mathcal{T}$. To choose the transition of $\mathcal{T}$, she can use the resolver r: given the run ρ built from the delays (including d_i) and transitions played so far, she plays $r(\rho, (a_i, b_i))$. Since r is a resolver, this strategy guarantees that the resulting run is accepting, and hence that she wins the synthesis game on $\mathcal{T}'$.

On the other hand, if Player 1 wins the synthesis game on $\mathcal{T}$, he has a strategy s which guarantees a play $w \in (\Sigma_i \times \Sigma_O)^T$ that is not in the language of $\mathcal{T}$. He can use the same strategy in the synthesis game of $\mathcal{T}'$ to guarantee a play (w, ρ) such that w is not in the language of $\mathcal{T}$, and by extension (w, ρ) is not in the language of $\mathcal{T}'$, as there are no accepting runs over w in $\mathcal{T}$.

The lower bound follows from the ExpTime-completeness of synthesis for deterministic TA [DM02]. □

8. Conclusion

We introduced history-determinism for timed automata and showed that several natural decision problems – timed language inclusion, universality and synthesis – that are undecidable in general, remain decidable for such automata. We proved that for the important subclasses of timed safety and timed reachability automata, history-determinism can be checked (and therefore good-enough synthesis of deterministic reachability and safety automata can be solved) in exponential time.

We also showed that every history-deterministic timed safety or reachability automaton can be determinized, based on pruning the region automaton, and therefore these classes of automata are strictly less expressive than their fully non-deterministic counterparts.

We further establish that in terms of recognizing languages over infinite timed words, HD timed automata are strictly in between fully deterministic and the more general non-deterministic TA. More precisely, we present a history-deterministic one-clock coBüchi timed automata whose language is not recognized by any k-clock deterministic parity TA, and a non-deterministic 1-clock reachability TA whose language is not recognized by any HD k-clock parity TA.

We leave open the comparative expressiveness of history-deterministic Büchi timed automata: are they determinizable? Already in the untimed case [ARK19], positional resolvers for HD Büchi may not exist, but a finite amount of memory suffices. We conjecture that HD Büchi timed automata can still be determinized via resolvers that are based on regions and a finite amount of additional memory.

Let us conclude with another conjecture. We showed that history-deterministic timed automata are "good" for solving turn-based timed games, where in each turn of the game, one of the two players chooses a time delay or an action. A more general, concurrent setting for timed games is presented in [dAFH$^+$03]. In their concurrent version, both players simultaneously choose permissible pairs of time delays and actions, and the player who has picked the shorter time delay gets to move. While concurrent games may not be determined, we conjecture that these concurrent timed games can again be solved by composing the (timed) arena with the (timed) winning condition, as long as the winning condition is history-deterministic.

References

[AD94]   Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 1994. `doi:10.1016/0304-3975(94)90010-8`.

[AFH99]   Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 1999. `doi:10.1016/S0304-3975(97)00173-4`.

[AH92]   Rajeev Alur and Thomas A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *33rd Annual Symposium on Foundations of Computer Science*, 1992. `doi:10.1109/SFCS.1992.267774`.

[AK20]   Shaull Almagor and Orna Kupferman. Good-enough synthesis. In *Computer Aided Verification (CAV)*, 2020. `doi:10.1007/978-3-030-53291-8_28`.

[ARK19]   Bader Abu Radi and Orna Kupferman. Minimizing gfg transition-based automata. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2019. `doi:10.4230/LIPIcs.ICALP.2019.100`.

[BHL$^+$22] Sougata Bose, Thomas A. Henzinger, Karoliina Lehtinen, Sven Schewe, and Patrick Totzke. History-deterministic timed automata are not determinizable. In *International Workshop on Reachability Problems (RP)*, 2022. `doi:10.1007/978-3-031-19135-0_5`.

[BK18]   Marc Bagnol and Denis Kuperberg. Büchi Good-for-Games Automata Are Efficiently Recognizable. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2018. `doi:10.4230/LIPIcs.FSTTCS.2018.16`.

[BKLS20] Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michal Skrzypczak. On succinctness and recognisability of alternating good-for-games automata. *CoRR*, 2020. `arXiv:2002.07278`.

[BL19]   Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In *International Conference on Concurrency Theory (CONCUR)*, 2019. `doi:10.4230/LIPIcs.CONCUR.2019.19`.

[BL21]   Udi Boker and Karoliina Lehtinen. History Determinism vs. Good for Gameness in Quantitative Automata. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2021. `doi:10.4230/LIPIcs.FSTTCS.2021.38`.

[BL22]   Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative automata. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2022. `doi:10.1007/978-3-030-99253-8_7`.

[BPDG98] Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 1998. `doi:10.3233/FI-1998-36233`.

[CHP08]   Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2008. `doi:10.1007/978-3-540-85778-5_10`.

[Col09]   Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2009. `doi:10.1007/978-3-642-02930-1_12`.

[dAFH+03]   Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In *International Conference on Concurrency Theory (CONCUR)*, 2003. `doi:10.1007/978-3-540-45187-7_9`.

[DJW97]   S. Dziembowski, M. Jurdziński, and I. Walukiewicz. How much memory is needed to win infinite games? In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 1997. `doi:10.1109/LICS.1997.614939`.

[DM02]   Deepak D'souza and P. Madhusudan. Timed control synthesis for external specifications. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2002. `doi:10.1007/3-540-45841-7_47`.

[Fin06]   Olivier Finkel. Undecidable problems about timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2006. `doi:10.1007/11867340_14`.

[FLW20]   Emmanuel Filiot, Christof Löding, and Sarah Winter. Synthesis from weighted specifications with partial domains over finite words. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2020. `doi:10.4230/LIPIcs.FSTTCS.2020.46`.

[GH82]   Y. Gurevich and L. Harrington. Trees, automata and games. In *Symposium on Theory of Computing (STOC)*, 1982. `doi:10.1145/800070.802177`.

[GJLZ21]   Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A Bit of Nondeterminism Makes Pushdown Automata Expressive and Succinct. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2021. `doi:10.4230/LIPIcs.MFCS.2021.53`.

[HKR97]   Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. In *International Conference on Concurrency Theory (CONCUR)*, 1997. `doi:10.1007/3-540-63141-0_19`.

[HKW95]   Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 1995. `doi:10.1007/3-540-60084-1_93`.

[HLT22]   Thomas A. Henzinger, Karoliina Lehtinen, and Patrick Totzke. History-deterministic timed automata. In *International Conference on Concurrency Theory (CONCUR)*, 2022. `doi:10.4230/LIPIcs.CONCUR.2022.14`.

[HNSY92]   T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 1992. `doi:10.1109/LICS.1992.185551`.

[HP06]   Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *Computer Science Logic (CSL)*, 2006. `doi:10.1007/11874683_26`.

[JLS08]   Marcin Jurdzinski, Francois Laroussinie, and Jeremy Sproston. Model Checking Probabilistic Timed Automata with One or Two Clocks. *Logical Methods in Computer Science*, 2008. `doi:10.2168/LMCS-4(3:12)2008`.

[KS15]   Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2015. `doi:10.1007/978-3-662-47666-6_24`.

[KSV06]   Orna Kupferman, Shmuel Safra, and Moshe Y Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 2006. `doi:10.1016/j.apal.2005.06.009`.

[LZ22]   Karoliina Lehtinen and Martin Zimmermann. Good-for-games ω-pushdown automata. *Logical Methods in Computer Science*, 2022. `doi:10.46298/lmcs-18(1:3)2022`.

[Mar75]   Donald A Martin. Borel determinacy. *Annals of Mathematics*, 1975. `doi:10.2307/1971035`.

[Sch20]   Sven Schewe. Minimising good-for-games automata is np-complete. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2020. `doi:10.4230/LIPIcs.FSTTCS.2020.56`.

[TAKB96]   Serdar Tasiran, Rajeev Alur, Robert P. Kurshan, and Robert K. Brayton. Verifying abstractions of timed systems. In *International Conference on Concurrency Theory (CONCUR)*, 1996. `doi:10.1007/3-540-61604-7_75`.