

On History-Deterministic One-Counter Nets

Aditya Prakash¹ and K. S. Thejaswini¹

Department of Computer Science, University of Warwick
`{aditya.prakash,thejaswini.raghavan.1}@warwick.ac.uk`

Abstract. We consider the model of history-deterministic one-counter nets (OCNs). History-determinism is a property of transition systems that allows for a limited kind of non-determinism which can be resolved ‘on-the-fly’. Token games, which have been used to characterise history-determinism over various models, also characterise history-determinism over OCNs. By reducing 1-token games to simulation games, we are able to show that checking for history-determinism of OCNs is decidable. Moreover, we prove that this problem is **PSPACE**-complete for a unary encoding of transitions, and **EXPSpace**-complete for a binary encoding.

We then study the language properties of history-deterministic OCNs. We show that the resolvers of non-determinism for history-deterministic OCNs are eventually periodic. As a consequence, for a given history-deterministic OCN, we construct a language equivalent deterministic one-counter automaton. We also show the decidability of comparing languages of history-deterministic OCNs, such as language inclusion and language universality.

Keywords: History-determinism · Token games · One-counter nets · One-counter automaton.

1 Introduction

While deterministic automata are algorithmically efficient for problems such as synthesis or for solving games, they are often much less succinct, or less expressive than their non-deterministic counterparts. The notion of history-determinism was introduced by Henzinger and Piterman [15] for automata over infinite words with parity acceptance conditions, as a tool to solve synthesis games efficiently. Such automata are known to compose well with games, and hence are also called good-for-games (GFG) automata [15,11]. History-deterministic automata form a robust class of models that is both algorithmically and conceptually interesting, and has been extensively studied over the recent years [15,11,4,25,6,1,5,9,28].

The notion of history-determinism emerged independently in the setting of cost automata, that can capture all regular cost functions as opposed to their deterministic version [10]. Recently, history-determinism has been studied in other quantitative settings [7,8], as well as infinite-state systems such as pushdown automata [13,26], Parikh automata [12], and timed automata [14].

One-counter nets are finite-state systems along with a counter that stores a non-negative integer value that can never be explicitly tested for zero. They correspond to 1-dimensional VASS, Petri nets with exactly one unbounded place, and are a subclass of one-counter automata which do not have zero tests, and hence are also a subclass of pushdown automata. They are one of the simplest infinite-state systems, and hence many problems pertaining to one-counter nets are easier than their counterparts that subsume them.

The structure of the resolvers that resolve non-determinism on-the-fly are crucial to understand history-determinism in various models. While for automata over infinite words with parity conditions, these resolvers take the shape of deterministic parity automata [15], the situation for resolvers in history-deterministic infinite-state systems is not as well understood. Indeed, the computability of such a resolver for a given history-deterministic pushdown automaton is left as an open problem in the works of Guha, Jecker, Lehtinen and Zimmermann [13]. For history-deterministic Parikh automata, it is still an open problem if the resolver can be given by a deterministic Parikh transducer [12]. Moreover, many other problems such as deciding history-determinism or even language inclusion among history-deterministic automata are undecidable for pushdown automata and Parikh automata [13,26,12]. We consider history-determinism over a well-studied class of infinite-state systems of one-counter nets, where we are able to answer positively to all of the above questions.

The techniques we use to answer several of these questions use results and techniques from the simulation problem over one-counter nets [17,16]. This is not surprising, since simulation of various models has close ties with history-determinism [15,14].

Our Contribution We study history-deterministic OCNs and establish them as a class of infinite-state systems where many problems pertaining to history-determinism are decidable. This is unlike other classes of history-deterministic infinite-state systems that subsume them.

Firstly, we show that checking for history-determinism for a given OCN is **PSPACE**-complete when transitions are encoded in unary, and is **EXSPACE**-complete for a succinct encoding (Theorem 4, Theorem 27). We achieve the upper bound by giving a novel reduction from the 1-token game G_1 to the simulation problem over OCNs. 1-token games characterise history-determinism over OCNs, and thus our reduction further extends the link between history-determinism and simulation. This decidability result is in contrast to one-counter automata (OCA), where checking for history-determinism becomes undecidable by just adding zero-tests to OCNs (Theorem 28).

Secondly, we show that resolvers for non-determinism in history-deterministic OCNs can be expressed as an eventually periodic set. Using this, we are able to determine history-deterministic OCNs to give a language equivalent deterministic OCA.

Finally, we show the decidability of the problems of language inclusion and language universality for history-deterministic OCNs to be in **PSPACE** and **P** respectively. This is unlike non-deterministic OCNs, where these problems

are known to be undecidable and Ackermann-complete respectively. Even for the class of deterministic OCA, which we show history-deterministic OCNs can be converted to, the inclusion problem is known to be undecidable.

Organisation of the paper Section 2 contains preliminaries where we introduce notation and define the concepts mentioned above rigorously. In Section 3 we show **PSPACE**-completeness of checking if an input OCN is history-deterministic. In Section 4, we show that the language expressed by history-deterministic one-counter nets are contained in the language accepted by deterministic one-counter automata. Moreover, we discuss the complexity of checking language-inclusion, language-equivalence and universality of history-deterministic nets. Finally, in the Section 5, we analyse the changes in complexity when the counters are represented succinctly, or if zero tests are added. Due to space constraints, missing proofs can be found in the appendix.

2 Preliminaries

We use Σ throughout to denote a finite set of alphabet, and Σ^* to denote the set of all finite words consisting of letters from Σ . The empty word over Σ shall be denoted by ϵ . We use Σ_ϵ to denote the set $\Sigma \cup \{\epsilon\}$. A language \mathcal{L} over Σ is a subset of Σ^* .

Labelled Transition System A *labelled transition system* (LTS) is a tuple \mathcal{S} consisting of $\mathcal{S} = (Q, \Sigma_\epsilon, \rightarrow, q_0, F)$. In this paper, we assume Q to be a (countable) set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is the set of final states, Σ is a finite alphabet, $\rightarrow \subseteq Q \times \Sigma_\epsilon \times Q$ is the set of transitions.

If a transition (q_1, a, q_2) belongs to \rightarrow , we instead represent it as $q_1 \xrightarrow{a} q_2$ as well. On a (finite) word w , a ρ is said to be a (finite) *run* of the labelled transition system \mathcal{A} if it is an (finite) alternating sequence of states and letters of Σ : $\rho = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \dots q_{k-1} \xrightarrow{a_{k-1}} q_k$, where each $q_i \xrightarrow{a_i} q_{i+1} \in \rightarrow$ and $a_0 \cdot a_1 \dots a_k = w$ and $a_i \in \Sigma_\epsilon$. A run ρ described above is accepting if the state $q_k \in F$.

An LTS that has no ϵ -transitions is said to be a *realtime LTS*. For $\mathcal{S} = (Q, \Sigma, \rightarrow, q_0, F)$ being realtime, we have $\rightarrow \subseteq Q \times \Sigma \times Q$. Unless mentioned otherwise, we mostly deal with realtime LTS for the sake of a simpler presentation, although our results also holds for systems with ϵ -transitions.

An LTS $\mathcal{S} = (Q, \Sigma, \rightarrow, q_0, F)$ is *deterministic* if \rightarrow is a function from $Q \times \Sigma$ to Q , and not just a relation.

Two player games Throughout the paper, we will be using two player games on countably sized arenas, between the players Adam and Eve, denoted by \forall and \exists respectively. The winning condition will be a reachability condition for one of the players, often \forall . By the work of Martin [27], we know that such games are determined, that is they have a winner, which is either \forall or \exists . Moreover, each of the players have a positional strategy, where their current strategy depends on

their positions in the current arena. We shall say that two games are *equivalent*, if they have the same winner.

One-Counter Automata A *one-counter automaton* (OCA) \mathcal{A} is given by a tuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, Σ is a finite alphabet, and finally, Δ is the set of transitions, given as a relation $\Delta \subseteq Q \times \{\text{zero}, \neg\text{zero}\} \times \Sigma \times \{-1, 0, 1\} \times Q$.

Here, the symbols *zero* and $\neg\text{zero}$ are used to distinguish between transitions that can happen when the counter value is 0, and when the counter value is positive respectively. One can think of the counter as a ‘stack’, where the stack has a distinguished bottom-of-the-stack symbol. The configurations in the automaton are given by pairs (q, m) , where q denotes the current state, and $m \in \mathbb{N}_0$ denotes the counter value. We use $\mathcal{C}(\mathcal{A})$ to denote the set of configurations of \mathcal{A} .

A one-counter automaton can be viewed as a succinct description of an infinite-state LTS over the set of configurations, such that the configurations are as defined below. For each configuration (q, m) , upon reading $a \in \Sigma_\epsilon$,

- if $m > 0$, takes a transition of the form $(q, \neg\text{zero}, a, d, q')$, where $d \in \{-1, 0, 1\}$ to $(q', m + d)$;
- if $m = 0$, takes a transition of the form $(q, \text{zero}, a, d, q')$, where $i \in \{0, 1\}$ to $(q', m + d)$.

For two configurations $c, c' \in \mathcal{C}(\mathcal{A}) = Q \times \mathbb{N}_0$, we use the notation $c \xrightarrow{a,d} c'$ to denote the fact that c' can be reached from c upon taking some transition $\delta \in \Delta$ upon reading a , with a change of counter value d . We shall also say that $c \xrightarrow{a,d} c'$ is a transition in \mathcal{A} , as $c \xrightarrow{a,d} c'$ is a transition in the infinite LTS of \mathcal{A} . We thus view \mathcal{A} as both an automaton and a LTS, and switch between these two notions interchangeably. A run of \mathcal{A} over a word w is a finite sequence of alternating configurations and transitions : $\rho = c_0 \xrightarrow{a_0, d_0} c_1 \cdots c_n \xrightarrow{a_n, d_n} c_{n+1}$ such that $a_0 a_1 \cdots a_n = w$, and $c_0 = (q_0, 0)$. The run ρ is an *accepting run* if its last configuration $c_{n+1} = (q_{n+1}, k_{n+1})$ is accepting, i.e. $q_{n+1} \in F$. We say a word w is an *accepting word* in \mathcal{A} if it has an accepting run in \mathcal{A} . Finally, we define the language of \mathcal{A} , denoted by $\mathcal{L}(\mathcal{A})$ to be the set of all accepting words in \mathcal{A} . We say that a one-counter automaton \mathcal{A} is a *deterministic one-counter automaton*, if Δ is a (partial) function from $Q \times \{\text{zero}, \neg\text{zero}\} \times \Sigma$ to $\{-1, 0, 1\} \times Q$.

One-counter nets The model of *one-counter nets* (OCNs) can be interpreted as a restriction added to one-counter automaton that do not have the ability to test for zero. Alternatively, one can view this as a finite-state automaton that has access to a stack which can store only one symbol and no bottom-of-the-stack element. Any feasible run cannot pop an empty stack. More formally, a one-counter net \mathcal{N} is a tuple $(Q, \Sigma, \Delta, q_0, F)$ where Q is the set of finite states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final or accepting states. The set $\Delta \subseteq Q \times \Sigma \times \{-1, 0, 1\} \times Q$ are the transitions in the net \mathcal{N} .

The configurations of an OCN are similar to that of an OCA. It consists of a pair $(q, n) \in Q \times \mathbb{N}$. We shall use the notation $\mathcal{C}(\mathcal{N}) = Q \times \mathbb{N}$ to denote the set of configurations of \mathcal{N} . From a configuration (q, n) , we reach a configuration $(p, n + d)$ in one step, if there is a transition $\delta = (q, a, d, p)$, for some $a \in \Sigma$ and $d \in \{-1, 0, +1\}$ and $n + d \geq 0$. We can define a run on an OCN, an accepting run and an accepting word similar to an OCA.

Remark 1. For the most of the paper we talk about one-counter nets (automata) with unary transitions, i.e. transitions that increment or decrement the counter by at most 1. However, they are as expressive as succinct models where the transitions are given in binary. This can be observed, for instance, by giving a construction similar to that of Valiant's for deterministic pushdown automata (Section 1.7, [30]).

We also assume, unless stated otherwise that the nets \mathcal{N} we talk about are *complete*, that is from any configuration $c \in \mathcal{C}(\mathcal{N})$, we can always take a transition on reading $a \in \Sigma$. This can be ensured by adding a rejecting sink state, and adding transition on all the letters that do not modify the counter from each state, (including the sink state) to this sink state.

History-Deterministic One-Counter Nets We define history-determinism in the setting of one-counter net. We say an OCN \mathcal{N} is *history-deterministic*, if the non-deterministic choices required to accept a word w which is in $\mathcal{L}(\mathcal{N})$ can be made on-the-fly. These choices depend only on the word read so far, and do not require the knowledge of the future of the word to construct an accepting run for a word in $\mathcal{L}(\mathcal{N})$ (hence the term history-determinism). Formally, we say an OCN \mathcal{N} is history-deterministic, if \exists wins the letter game on \mathcal{N} defined below.

Definition 2 (Letter game for OCN). *Given an OCN $\mathcal{N} = (Q, \Sigma, \Delta, q_0, F)$, the letter game on \mathcal{N} is defined between the players \forall and \exists as follows: the positions of the game are $\mathcal{C}(\mathcal{N}) \times \Sigma^*$, with the initial position $((q_0, 0), \epsilon)$. At round i of the play, where the position is (c_i, w_i) :*

- \forall selects $a_i \in \Sigma$
- \exists selects a transition δ which can be taken at the configuration c_i on reading a_i , i.e. $c_i \xrightarrow{a_i, d_i} c_{i+1}$

If \exists is unable to choose a transition, and $w_{i+1} = w_i a_i$ is the prefix of an accepting word, \exists loses immediately. The player \forall wins immediately when the word w_{i+1} is accepting but the configuration c_{i+1} is not at an accepting state, and the game terminates. The game continues from (c_{i+1}, w_{i+1}) otherwise. The player \exists wins any infinite play.

We say a strategy for \exists in the letter game of \mathcal{N} is a *resolver* for \mathcal{N} , if it is a winning strategy for \exists in the letter game.

Our characterization of history-deterministic one-counter net by the above letter game is slightly different from the one presented in the work of Guha, Jecker, Lehtinen and Zimmermann [13] for pushdown automata. In their work,

they define history-determinism as having a consistent strategy based on the transitions taken so far. It is easy to argue that they are equivalent.

The letter game can be formulated as a reachability game over countably many vertices, where the player \forall is trying to reach a position of the form $(c, w) \in \mathcal{C}(\mathcal{N}) \times \Sigma^*$, where c is at a rejecting state, while w is accepting. Such games are determined, and this follows from Martin's Theorem [27] showing that history-determinism formulated as a letter game is well-defined.

Letter games have been used in the literature to characterise history-determinism for other models as well, such as parity automata [15] and for various kinds of quantitative automata on both finite and infinite words [5,7].

To aid our understanding of history-determinism as well as the above definition, we provide an example of a game where \exists wins the letter game on this automaton but the strategy is based on her counter configuration.

Example 3. Consider the language

$$\mathcal{L} = \{a^n \$b^{n_1} \$b^{n_2} \$ \dots \$b^{n_k} \$ \mid \sum_{i=1}^k n_i \leq n \text{ or } n_k = 2, \sum_{i=1}^{k-1} n_i = n - 1\}.$$

which can be accepted by a history-deterministic OCN as shown in Figure 1. The initial state is indicated with an arrow pointing to it, and the final states are circled twice. Missing transitions are assumed to go to a rejecting sink state. In the corresponding letter game, \forall plays the letter a several times, say n -many times followed by a $\$$. The corresponding transitions so far are deterministic. Later, \forall reads some series of b s and $\$$ s, such that the word continues to be in the language. Note that the non-determinism occurs in only one state, on the letter b . A winning strategy of \exists which proves that this net is history-deterministic is the following: she takes the ‘down’ transition if the counter value is strictly larger than 1, but the ‘right’ transition on b otherwise. This non-determinism can’t be determined by removing transitions, because removing either of the ‘down’ b -transition or the ‘right’ b -transition changes the language accepted.

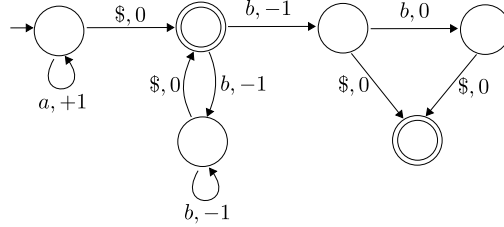


Fig. 1. A history-deterministic OCN accepting \mathcal{L}

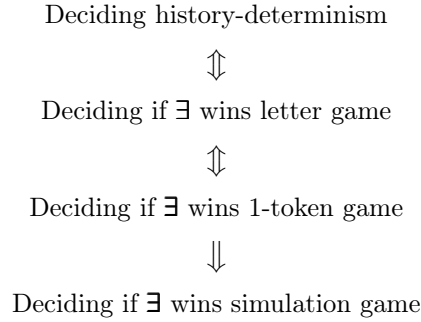
3 Deciding History-Determinism

The main result of this section is that deciding history-determinism for a given OCN is decidable and is **PSPACE**-complete as stated in the theorem below.

Theorem 4. *Given a one-counter net \mathcal{N} , checking if \mathcal{N} is history-deterministic is **PSPACE**-complete.*

The rest of this section is dedicated to the proof of the above statement.

The proof of showing the upper bound proceeds by a series of polynomial reductions as below.



We define these games rigorously and prove these reductions in Subsection 3.1. Finally, since the simulation problem for one-counter nets is **PSPACE** [16], this gives us the upper bound.

For the lower bound, we reduce the problem of emptiness checking for alternating finite-state automata over a unary alphabet to deciding if \exists wins the letter game.

3.1 Token games

Deciding history-determinism efficiently for finite-state parity automata over infinite words has been a major area of study over the recent years. Bagnol and Kuperberg, in their work [1], gave a polynomial time procedure for deciding history-determinism when the finite automata accepts with a Büchi condition. Their underlying technique is a two-player game, called G_2 or 2-token games, which they proved to be equivalent to the letter game when the automaton is Büchi. Boker, Kuperberg, Lehtinen and Skrzypczak [5] extended this to show that the game G_2 is equivalent to the letter game when the automaton is co-Büchi as well. Deciding the winner in G_2 for an automaton of a fixed parity index takes polynomial time [5], and hence deciding history-determinism for the cases of when the parity automata accepts words based on Büchi or co-Büchi condition is polynomial. It is famously conjectured [5] that winning G_2 is equivalent to the letter game for higher parity indices as well, and this is known as the G_2 conjecture. Token games have also been instrumental in deciding history-determinism for quantitative automata, in the works of Boker and Lehtinen [8]. In their paper, they show that for finite words on a finite-state

boolean automaton, history-determinism is characterised by G_1 . This was later extended to labelled transition systems with countably many states, in the works of Henzinger, Lehtinen and Totzke [14]. Thus, the 1-token games also characterise history-determinism over OCNs. We include a proof nonetheless, for the sake of completion.

In a play of the letter game, \forall picks the letters while \exists picks the transitions and the winning condition for \exists is to produce an accepting run for any word that is in the language. Token games work similarly, but they impose more restrictions on \forall . This is done by asking him to also display a valid run during the game with the help of some number of tokens. Here, we concentrate on the 1-token game G_1 . The player \forall wins the game G_1 if and only if he produces an accepting run, whilst \exists produces a rejecting run. We make this more formal in the definition below.

Definition 5 (One token game G_1). Let $\mathcal{N} = (Q, \Sigma, \Delta, q_0, F)$ be a one-counter net. The positions of the game G_1 on \mathcal{N} are a pair of configurations, $\mathcal{C}(\mathcal{N}) \times \mathcal{C}(\mathcal{N})$, where the first configuration is \exists 's token, and the second is \forall 's token. The game starts with the initial position $((q_0, 0), (q_0, 0))$. At the i^{th} iteration of the play, where the position is $((q_i^\exists, k_i^\exists), (q_i^\forall, k_i^\forall))$:

1. \forall selects $a \in \Sigma$
2. \exists selects a transition for her token, $(q_i^\exists, k_i^\exists) \xrightarrow{a,d} (q_{i+1}^\exists, k_{i+1}^\exists)$
3. \forall selects a transition for his token, $(q_i^\forall, k_i^\forall) \xrightarrow{a,d'} (q_{i+1}^\forall, k_{i+1}^\forall)$

The game goes to $((q_{i+1}^\exists, k_{i+1}^\exists), (q_{i+1}^\forall, k_{i+1}^\forall))$. If q_{i+1}^\forall is accepting but q_{i+1}^\exists is rejecting, \exists loses immediately and the game terminates. The player \exists wins any infinite play.

We show in the following lemma that \forall , even with limited power, in one-token games can capture letter games. Letter games can be seen as a version of token games where \forall plays with infinitely many tokens.

Lemma 6. For a OCN \mathcal{N} , if \exists wins the game G_1 on \mathcal{N} , then \exists has a winning strategy in the letter game as well.

To prove the above lemma, we need to understand better the structure of the resolvers for OCNs. Consider the definition given below of *residual transitions*. Intuitively, these are transitions such that if there was an accepting run from a state with the first letter as a , on taking a residual transition on a , then there is still an extension of the run from the new state. More formally, we say that a transition $(q, k) \xrightarrow{a,d} (q', k')$ is *residual* if $\mathcal{L}(q', k') = a^{-1}\mathcal{L}(q, k)$, where $\mathcal{L}(q, k)$ (and $\mathcal{L}(q', k')$) is the set of words that are accepted in \mathcal{N} when the initial configuration is (q, k) ((q', k')), instead of $(q_0, 0)$. The proposition below shows any winning strategy of \exists can be characterised by these residual transitions.

Proposition 7. For an OCN \mathcal{N} , an \exists strategy σ in the letter game is winning for \exists if and only if σ takes only residual transitions.

Note that in the letter game, each player winning the game has a positional winning strategy, as it is a reachability game. Suppose that \exists wins the letter game, then \exists has a winning strategy which can be given by a (partial) function $\sigma : (Q \times \mathbb{N}) \times \Sigma^* \times \Sigma \rightarrow \Delta^*$. Using Proposition 7, we can show that \exists 's strategy only depends on the configuration, and is independent of the word read so far.

Proposition 8. *If \exists wins the letter game, then \exists has a winning strategy σ that only depends on the current configuration of the play, i.e σ is a partial function $\sigma : (Q \times \mathbb{N}) \times \Sigma \rightarrow \Delta^*$*

Having shown that G_1 is equivalent to the letter game, we show that deciding the winner in the game G_1 is decidable in **PSPACE** (when the transitions are unary). This implies deciding history-determinism is decidable, and in **PSPACE**. We do so by reducing our problem to the simulation problem between two one-counter nets, which is known to be **PSPACE**-complete (cf. Theorem 7, [16]). Given two one-counter nets \mathcal{N} and \mathcal{N}' at configurations (q, n) and (q', n') , intuitively, we say \mathcal{N} simulates \mathcal{N}' from their corresponding configurations if for any sequence of transitions from (q, n) , there is also a sequence of transitions from (q', n') which is built ‘on-the-fly’. This alternation between existential and universal quantifiers in the above statement renders this definition perfect to be captured by the following simulation game between two players \forall and \exists .

Definition 9 (Simulation Game). *Given two (not necessarily complete) one-counter nets $\mathcal{N} = (Q, \Sigma, \Delta, F)$ and $\mathcal{N}' = (Q', \Sigma, \Delta', F')$ and two configuration $c = (p, k)$ and $c' = (p', k')$ in $\mathcal{C}(\mathcal{N})$ and $\mathcal{C}(\mathcal{N}')$ respectively where $k, k' \in \mathbb{N}$. The simulation game between the OCNs \mathcal{N} and \mathcal{N}' at a position (c, c') , denoted by $\mathcal{G}((\mathcal{N}, c) \hookrightarrow (\mathcal{N}', c'))$, is a two player game between \forall and \exists , with positions in $\mathcal{C}(\mathcal{N}) \times \mathcal{C}(\mathcal{N}')$ where the initial position is $(c_0, c'_0) = (c, c')$. At round i of the play, where the position is (c_i, c'_i) :*

- \forall selects a letter $a \in \Sigma$, and a transition $c_i \xrightarrow{a, d} c_{i+1}$ in \mathcal{N}
- \exists selects an a -transition $c'_i \xrightarrow{a, d'} c'_{i+1}$ in \mathcal{N}'

If \forall is unable to pick a transition, then \forall loses the game immediately. If \exists is unable to pick a transition after \forall has picked a transition, then \exists loses the game.

Otherwise, if \exists 's state in c'_{i+1} is rejecting, but \forall 's state in c_{i+1} is accepting, then \exists loses the game, and the game terminates. Else, the game goes to (c_{i+1}, c'_{i+1}) for another round of the play. The player \exists wins any infinite play.

If \exists wins the above game, we say $(\mathcal{N}, (p, k))$ simulates $(\mathcal{N}', (p', k'))$, and we denote it by $(\mathcal{N}, (p, k)) \hookrightarrow (\mathcal{N}', (p', k'))$. Furthermore, we say \mathcal{N} simulates \mathcal{N}' or $\mathcal{N} \hookrightarrow \mathcal{N}'$ if $(\mathcal{N}, (q_I, 0)) \hookrightarrow (\mathcal{N}', (q'_I, 0))$.

As the simulation game is a reachability game over a countably sized arena, it is determined, and the winning player has a positional strategy. Thus, if \exists wins the above simulation game $\mathcal{G}(\mathcal{N}, (p, k)) \hookrightarrow \mathcal{N}', (p', k'))$, then \exists has a positional winning strategy $\sigma_{\exists} : \mathcal{C}(\mathcal{N}) \times \mathcal{C}(\mathcal{N}') \times \Sigma \rightarrow \Delta'$.

Remark 10. In the literature over one-counter nets [29,16,22], the winning condition for the players on the simulation game is expressed differently, via the inability of the players to choose transitions, rather than accepting states. The player \forall (\exists) loses the game if \forall (\exists) is unable to choose a transition. It can however, be shown that the two versions of the simulation games are log-space reducible to each other. We show this equivalence in Appendix A.1.

Note the similarities (and differences) in G_1 and the simulation game. In both, the winning condition for \forall would like \forall 's run to be accepting, while \exists 's to be rejecting. In G_1 however, \exists is picking the transition first, while in the simulation game, \forall is picking the transition first.

With some modifications to the structure of the underlying nets in G_1 , we can ensure that the simulation game between the modified net and the original net captures G_1 . The intuition is that, in the simulation game, the net which simulates is modified such that \forall is forced to delay choosing his transition. This is formalized in the proof of the following lemma, and explained with a diagram in Figure 2.

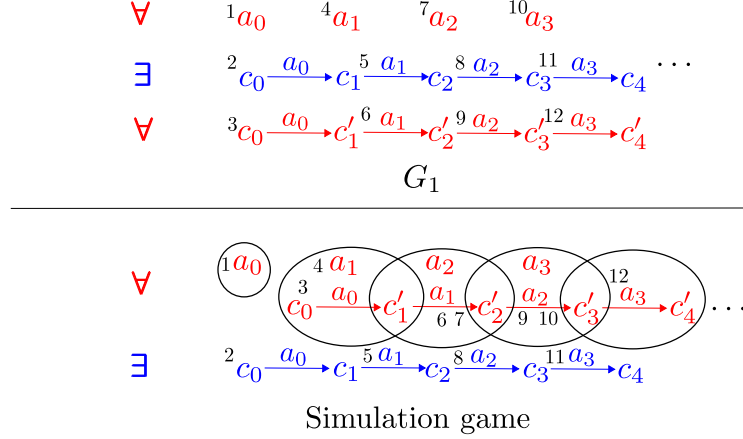


Fig. 2. An illustration of a play of G_1 , seen as a play of the simulation game

Lemma 11. *Given a one-counter net \mathcal{N} , there are one-counter nets \mathcal{M} and \mathcal{M}' , which have size at most polynomial in size of \mathcal{N} such that \exists wins G_1 on \mathcal{N} if and only if \exists wins $\mathcal{M} \hookrightarrow \mathcal{M}'$.*

Proof. (Sketch) Figure 2 captures the intuition behind the proof. Here note that we have different linearisations of the play, but the alternation between \exists and \forall required is captured by the simulation game by making \forall choose his configuration and transition at the same time. For each run in \mathcal{N} , we have a run in \mathcal{M} that lags behind one transition, and it does so by remembering which letter it should

move on next. We provide a construction such that \mathcal{M}' is linear in the size of \mathcal{N} and \mathcal{M} has size approximately $\mathcal{N} \times |\Sigma|$, where $|\Sigma|$ is the size of the alphabet. This factor of $|\Sigma|$ arises due to remembering in the state space, the previous letter read, to create a lag for \forall 's decisions. We then show that the player \exists wins G_1 on \mathcal{N} if and only if $\mathcal{M} \hookrightarrow \mathcal{M}'$.

Finally, we see that the following theorem from the work of Hofman, Lasota, Mayr and Totzke [16] shows that the winner of a simulation game can be solved in **PSPACE**. We recall their results to fit our notation below.

Theorem 12. *Given two one-counter nets \mathcal{N} and \mathcal{N}' , with configurations (p, k) and (p', k') in $\mathcal{C}(\mathcal{N})$ and $\mathcal{C}(\mathcal{N}')$ respectively, with k and k' represented in binary, deciding whether $(\mathcal{N}, (p, k))$ simulates $(\mathcal{N}', (p', k'))$ is in **PSPACE**. Moreover, the set of (k, k') for which $(\mathcal{N}, (p, k)) \hookrightarrow (\mathcal{N}', (p', k'))$ is semilinear, and can be computed in **EXPSPACE**.*

Proof. See [16], cf. Theorem 7

Lemma 13. *Given a one-counter net \mathcal{N} , we can decide in **PSPACE** if \mathcal{N} is history-deterministic.*

The above lemma is essentially a corollary of Lemma 6, Lemma 11 and Theorem 12.

3.2 Lower Bounds

Although solving the simulation game turns out to be **PSPACE**-complete itself from the work of Srba [29], this lower bound result does not work for our reduction to simulation games. The reduction we give from G_1 to simulation games produces only a restricted class of simulation games which solve G_1 .

Nevertheless, we show that deciding history-determinism is still **PSPACE**-hard, showing that even this restriction of the simulation problem is enough to induce **PSPACE**-hardness.

Lemma 14. *Given a one-counter net \mathcal{N} , it is **PSPACE**-hard to decide if \mathcal{N} is history-deterministic.*

Proof (Sketch). The proof goes by reducing from the problem of checking non-emptiness of an alternating finite-state automaton over a unary alphabet. This variation of the problem was proven to be **PSPACE** complete by Holzer [19], with its proof simplified by Jančar and Sawa [23]. The intuition behind the construction is to recreate a run of the alternating automaton using the constructed net. In the letter game, a fair play of \forall corresponds to a branch of a run-tree in the automaton, with \exists resolving universal transitions and \forall resolving existential ones. The player \forall can ensure that he wins the letter game if and only if the alternating automaton has some word that he can demonstrate is in the language. If \forall plays unfairly, then there are gadgets to ensure that \exists automatically wins.

We conclude this section by saying that Lemma 13 and Lemma 14 together give the proof of Theorem 4

4 Languages and History-Determinism in OCNs

. We dedicate this section to tackling different questions about languages of history-deterministic one-counter nets and decision problems of such languages.

4.1 Languages Accepted by History-Deterministic OCNs

While in history-deterministic models we are able to resolve the non-determinism on-the-fly, it is not well-understood how these resolvers might look like in general. In fact, Guha, Jecker, Lehtinen and Zimmermann showed that there are history-deterministic pushdown automata whose resolvers cannot be given by a pushdown automata [13], and whether such a resolver can be computed is an open problem.

In this sub-section, our goal is to understand better the language of history-deterministic OCNs. As a first-step towards this goal, we already have some intuition from the previous section on the eventually periodic nature of the transitions that are residual (as a corollary of Lemma 11 and Theorem 12). Here, we solidify this intuition by defining what it means to have *semilinear-strategy property* for a resolver and to then show that all nets have this property. For the case of history-deterministic nets, using this semi-linearity of the resolvers, we show the existence of a language-equivalent deterministic OCA.

We first show a sufficient characterisation which we call the *semilinear-strategy property*, for if a given history-deterministic one-counter net can be determined.

We say a transition $\delta = (p, a, d, p')$ in an one-counter net \mathcal{N} is a good transition at (p, k) , if $((p, k), (p, k))$ is in the winning region of G_1 , and the transition $\delta = (p, k) \xrightarrow{a, d} (p', k + d)$ is a winning move for \exists when \forall chooses the letter a . We also write this sometimes as $(p, k) \xrightarrow{a, d} (p', k + d)$ is a good transition in \mathcal{N} . The following lemma can be seen as a weakening of Proposition 7 :

Lemma 15. *Let $\mathcal{N} = (Q, \Sigma, \Delta, q_0, F)$ be a history-deterministic one-counter net. An \exists strategy σ in the letter game is winning for \exists if and only if the strategy σ only takes good transitions $\delta = (p, k) \xrightarrow{a, d} (p', k')$.*

Proof. Note that any strategy of \exists that is winning in the letter game takes only good transitions, as G_1 is a weaker game for \exists than the letter game. The other direction follows by observing that any good transition is also residual. If $(p, k) \xrightarrow{a, d} (p', k')$ is good, then for any word $aw \in L(p, k)$, we must have $w \in L(p', k')$: If not, then \forall can win G_1 by constructing an accepting run on aw from (p, k) which contradicts the definition of good transitions. Hence the proof follows from Proposition 7.

Definition 16. *Given a one-counter net \mathcal{N} , we say \mathcal{N} satisfies semilinear-strategy property if for each transition $\delta = (q, a, d, q')$, the set of $k \in \mathbb{N}$ such*

that δ is a good transition at (q, k) is semilinear. That is for each transition $\delta = (q, a, d, q') \in \Delta$, we have that the set

$$\mathcal{S}_\delta = \{k : (q, k) \xrightarrow{a, d} (q', k') \text{ is a good transition at } (q, k)\}$$

is semilinear.

Consider the following example which solidifies this intuition:

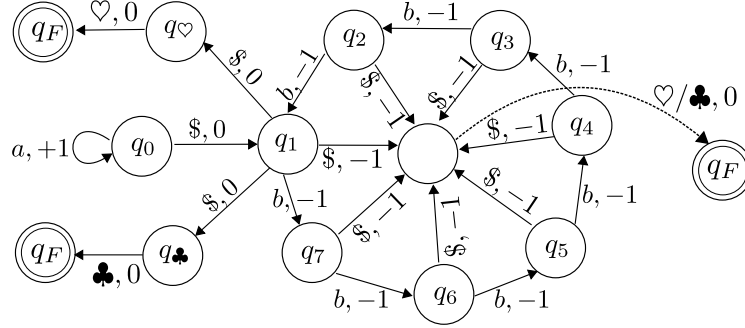


Fig. 3. The one-counter net \mathcal{N}_7 from Example 17

Example 17. Consider the net \mathcal{N}_7 , as shown in Figure 3, where all states labelled q_F are accepting. This automaton is not history-deterministic. However, if the counter value at q_1 is not a multiple of 7, then \exists can resolve the non-determinism from q_1 . Observe that the automaton accepts words of the form $a^n \$ b^k \$ \cdot (\heartsuit, \clubsuit)$ such that $k \leq n$. Consider the following play of \forall in the letter game from q_0 : For $7n$ steps he reads a , after which he reads a $\$$. So far, all transitions are deterministic. After that, assume he again reads, $7n$ many times, the letter b . This ensures that the transition ends at state q_1 with counter value 0. If he reads $\$$ here, this is the only position where \exists has a choice. Note that she has to choose between q_\heartsuit and q_\clubsuit and since both the suffix \heartsuit and \clubsuit are accepting, she loses no matter what she picks. However, if \forall had read a number of ‘ b ’s was not a multiple of 7, the play of an accepting word would end at $q_\$$ which is accepting. This serves to show two things: firstly, a non-example of history-determinism, and secondly, how the counter values affect the decisions of the player, which is in this case, \forall .

Lemma 18. *If a history-deterministic one-counter net $\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$ satisfies the semilinear-strategy property, then there is a language-equivalent deterministic OCA \mathcal{D} .*

Proof (Sketch). We assume the history-deterministic one-counter net \mathcal{N} is such that it satisfies semilinear-strategy property. We first construct a non-deterministic

one-counter automata \mathcal{B} . The non-deterministic one-counter automata \mathcal{B} would essentially be designed so that one can only take good transitions from any configuration. The eventual periodicity of the sets S_δ allows us to express this as a one-counter automaton, rather than as a labelled transition system with countably many states.

Intuitively, the automaton \mathcal{B} is constructed such that the state space of the automaton stores in its memory the period and the initial block of the semilinear sets. The idea is that this automaton's runs would be in bijection with those runs that take only good transitions in the OCN \mathcal{N} . We know that such a run exists in \mathcal{N} by Lemma 15, as \mathcal{N} is history-deterministic. However, the counter values in \mathcal{B} are 'scaled down' to only remember how many periods have passed, while counter value 0 indicates that the counter value in the original run would have been at most I . The exact value of the counter value in a run of \mathcal{N} can be inferred as a function of the state space.

Having shown that if a history-deterministic one-counter net satisfies semilinear-strategy property, then we have an equivalent DOCA, we proceed to show that every one-counter net satisfies semilinear-strategy property.

Lemma 19. *Every one-counter net \mathcal{N} satisfies semilinear-strategy property.*

The proof of the above lemma is similar to the proof of Lemma 11. As an easy corollary of the above two lemmas, we get the following theorem.

Theorem 20. *Every history-deterministic OCN can be determinised to produce an equivalent deterministic OCA.*

An easy analysis of our proof combined with the results on the representation of simulation preorder (Lemma 28, [16]) shows a doubly exponential upper bound on the size of the equivalent deterministic OCA constructed from the proof of the theorem above. However, we conjecture that there exists an (at most) exponentially sized language-equivalent deterministic OCA for every history-deterministic OCN.

Remark 21. On the topic of expressivity of history-determinism, we conclude this subsection with a remark that history-deterministic OCNs are strictly less expressive than their counterparts which are allowed to exhibit full non-determinism. This can be demonstrated with the following language

$$\mathcal{L} = \{a^i b^j b^k \mid j \leq i \text{ or } k \leq i\}.$$

It is routine to verify that such a language is not accepted by any history-deterministic automaton, but this language can be accepted by a non-deterministic OCN. Note that history-determinism itself is not the limiting factor in accepting this language, as this language is accepted by a history-deterministic pushdown automaton [13].

4.2 Complexity of comparing languages of history-deterministic OCNs

The complexity of comparisons between languages of non-deterministic one-counter nets are undecidable [17], and even the restricted question of universality, is Ackermann-complete [18]. Whereas for deterministic one-counter automata, although equivalence and therefore universality is in **NL** [2,3], inclusion is undecidable [30]. In this section, we show that for history-deterministic nets, these problems are no longer undecidable and have a significantly lower complexity when compared to non-deterministic nets.

Note that although we have a procedure to determinise our automaton earlier in this section, this procedure does not help us answer these questions. This is because our determinisation procedure results in a deterministic OCA rather than an deterministic OCN. For deterministic OCNs, all these problems are known to be **NL**-complete [18], but for deterministic OCA, the problem of inclusion is undecidable [30]. Even though equality and universality for a deterministic OCA is **NL** complete, the resulting deterministic OCA we get from determinisation of history-deterministic OCNs could be much larger than our input net, leading to much larger complexity.

Nevertheless, we show that checking language inclusion and hence checking language equivalence between two history-deterministic one-counter nets is in **PSPACE**. This is done by showing a reduction to the problem of deciding history-determinism. Recall that as this problem is in **PSPACE** from Lemma 13 and Theorem 4, we are able to show membership in **PSPACE** for language equivalence and inclusion between two history-deterministic one-counter nets. Moreover, using results of Kucera [24], we get decidability in **P** for language universality.

Lemma 22. *Deciding language inclusion and language equivalence between two history-deterministic one-counter nets is in **PSPACE**.*

We can show that the problem of checking language inclusion between two history-deterministic OCNs reduces to checking if a larger OCN (linear in the sum of the size of the two OCNs) is history-deterministic. Since language equivalence is essentially checking language inclusion both ways, we have the above results.

Lemma 23. *Deciding language universality for a given history-deterministic one-counter net is in **P**.*

The problem of universality reduces to checking if a finite-state automaton simulates the input net \mathcal{M} . This problem was shown to be **P** by Kucera (Lemma 2, [24]), showing that universality is in **P**.

We therefore have the following theorem.

Theorem 24. *For nets \mathcal{H} and \mathcal{H}' that are history-deterministic, the problem of checking if $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{H}')$ as well as checking if $\mathcal{L}(\mathcal{H}) = \mathcal{L}(\mathcal{H}')$ can be done in **PSPACE**. If \mathcal{H} is instead a deterministic finite-state automaton, this problem can be solved in **P**.*

We summarise known results and complexity of relevant results for comparison with other automata models in table 1.

	$\mathcal{L} \subseteq \mathcal{L}'$	$\mathcal{L} = \mathcal{L}'$	$\mathcal{L} = \Sigma^*$
DOCN	NL -complete [18]	NL -complete [18]	NL -complete [18]
HOCN	In PSPACE	In PSPACE	In P
OCN	Undecidable [30]	Undecidable [17]	Ackermann-complete [18]
DOCA	Undecidable [30]	NL -complete [2]	NL -complete [2]

Table 1. Complexity of deciding language inclusion, equivalence and universality over deterministic OCN, history-deterministic OCN, non-deterministic OCN and deterministic OCA.

5 Extensions and Variations of OCN

We revisit the question of deciding history-determinism in this section for one-counter nets and its variants. In the first subsection, we tackle the question of how the complexity changes if the encoding of these nets are given in binary. We show that as expected, this increases the complexity of the problem from **PSPACE**-complete to **EXSPACE**-complete. We then answer affirmatively to the question of whether adding zero-tests add too much power to one-counter nets by showing that the problem of deciding history-determinism becomes undecidable.

5.1 Succinct Encoding of Counters

If the input nets are such that the modifications to the counters are encoded in binary, we show that the problem of deciding history-determinism becomes **EXSPACE**-complete.

Suppose we allow for an increment and a decrement in our net which is encoded as a binary input, then we wish to see if deciding history-determinism in such nets is harder. Unsurprisingly, we can show that this problem takes **EXSPACE** when the nets are encoded in binary, which we remark in the following Lemma.

Proposition 25. *Given a net \mathcal{N} is such that transitions allow for binary encoding of the value, then deciding if \mathcal{N} is history-deterministic is in **EXSPACE**.*

This result follows from the previous proof of **PSPACE** upper bound from Lemma 13 of deciding history-determinism for one-counter nets, where counter values are in unary. Any net with binary encoding can be converted with only an exponential blow-up into another language equivalent net with unary encoding,

preserving history-determinism. This naturally gives us an **EXPSPACE** upper bound.

However, much more work is needed to show a matching lower bounds, which we do by giving a reduction from reachability games on succinct one-counter net (SOCN)s. Intuitively, these games are played on the configuration graphs of a one-letter OCN, where the states of the OCN are partitioned among two players, which we denote by \wedge and \vee . The goal of the \vee is to be able to take the play to a designated winning state with value 0. This problem was shown to be **EXPSPACE** complete by Hunter [20] and later, several of its variants were also shown to have the same complexity [22]. This therefore gives us our **EXPSPACE**-completeness for deciding history-determinism of one-counter nets. The name succinct comes from the encoding of the net in the input.

Lemma 26. *Given an OCN \mathcal{N} , where the numbers in the transitions are represented in binary, deciding if \mathcal{N} is history-deterministic is **EXPSPACE**-hard.*

Proof (Sketch). Given an instance of a SOCN-reachability game on \mathcal{N} , We construct an OCN \mathcal{M} such that \vee wins in the SOCN-reachability game on \mathcal{N} if and only if \forall wins in the letter game on \mathcal{M} .

The high-level idea of the construction is that we construct an automaton \mathcal{M} , such that in a play of the letter game on \mathcal{M} , the players \forall and \exists create a transcript of a run of the automaton \mathcal{N} . This is done easily by \forall picking the letters at \vee states, where he can pick a different letter, each corresponding to a different transition. Since in the letter game, we have in \exists to resolve the non-determinism, we do that to allow for \exists to resolve the choices of the \wedge player.

However, we need to ensure a few important aspects. Firstly, any transcript (word) created so far indeed corresponds to a valid run on the automaton \mathcal{M} , and that \forall respects \exists 's choice when she resolves non-determinism. These are the main challenges while constructing such an automaton and they are resolved by the use of a few gadgets that we describe in detail in the appendix.

We conclude this subsection with the following theorem.

Theorem 27. *Given an OCN \mathcal{N} where the numbers in the transitions are represented in binary, deciding if \mathcal{N} is history-deterministic is **EXPSPACE**-complete.*

5.2 Deciding History-Determinism for OCA

We show that, given a one-counter automaton \mathcal{A} , deciding if \mathcal{A} is history-deterministic is undecidable. It was shown by Guha, Jecker, Lehtinen and Zimmermann [13] that deciding if a non-deterministic pushdown automaton is history-deterministic is undecidable. This extends their result to OCAs. The reduction follows from the undecidability of language inclusion for deterministic one-counter automata [30].

Theorem 28. *Given an OCA \mathcal{A} , deciding if \mathcal{A} is history-deterministic is undecidable.*

Proof (Sketch). Consider the following problem :

DOCA Inclusion: Given two DOCAs \mathcal{A} and \mathcal{B} , is $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$?

The above problem was shown to be undecidable in Section 5.1 of Valiant’s thesis [30]. We show that the problem of deciding if a given one-counter automaton is history-deterministic is also undecidable, by the means of a reduction.

This shows that zero-tests already add too much power for the problem of deciding history-determinism.

6 Discussion

We showed several decision problems related to history-determinism to be decidable over OCNs. This is unlike other classes of infinite-state systems that subsume them, where either a subset or all of the problems are undecidable.

We note that we only deal with realtime nets with no ϵ -transitions, but our results hold without too much modification when ϵ -transitions are present, as weak simulation over OCNs can be decided in **PSPACE** (and **EXSPACE** for a succinct encoding), and the weak simulation pre-order is semilinear as well [16]. We considered some model-related variations and concluded that testing the counter for zero freely made checking for history-determinism undecidable. One could ask about models like reversal bounded one-counter automata [21], or automata with bounded number of zero-tests, to gauge the frontier between decidability and undecidability on these systems.

Although not obvious from the main part of the paper, we are confident that our results could easily be extended to safety acceptance conditions. One could also ask, for instance, to look at reachability or Büchi and co-Büchi acceptance conditions and understand how history-determinism works in these models.

There are several questions about the expressivity of history-deterministic OCNs which we believe need further study. We have shown that

$$\text{DOCN} \subseteq \text{HOCN} \subseteq \text{OCN} \cap \text{DOCA}.$$

An interesting problem would be to prove or disprove if any of these inclusions are strict. In fact, we don’t have an example of a language that is accepted by a history-deterministic OCN which is not accepted by a deterministic OCN.

One could ask similar questions about expressivity of history-determinism in OCAs, i.e. if $\text{HOCA} = \text{DOCA}$. Although deciding history-determinism is undecidable, it might be possible for one to show that the language accepted by a history-deterministic OCA is as expressive as deterministic OCA. We remark that the 1-token game G_1 characterises history-determinisation for OCAs as well. Moreover, we can again show with similar techniques that if history-deterministic OCAs satisfy the semilinear-strategy property, then these languages can also be

expressed by a deterministic OCA. The key part that we need to prove for determinisation of history-deterministic OCA would be the semilinear-strategy property. It would be interesting to see how such a proof would look like, given the status of deciding history-determinism being undecidable for OCA.

Acknowledgements We would like to thank Dmitry Chistikov for listening to our conjectures and pointing us to important references. We are also grateful for his comments on our introduction. We are thankful to Neha Rino for carefully proofreading our paper, and suggesting improvements in our presentation. We also thank Sougata Bose, Piotrek Hofman, Filip Mazowiecki, David Purser and Patrick Totzke for their insightful remarks on our draft, and for telling us about weak simulation. We are grateful to Shaul Almagor and Asaf Yeshurun for a fun talk about OCNs. Finally, we thank Marcin Jurdziński for his support, and for bringing us his homemade rhubarb crumble.

References

1. Bagnol, M., Kuperberg, D.: Büchi Good-for-Games Automata Are Efficiently Recognizable. In: Ganguly, S., Pandya, P. (eds.) 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018). Leibniz International Proceedings in Informatics (LIPIcs), vol. 122, pp. 16:1–16:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). <https://doi.org/10.4230/LIPIcs.FSTTCS.2018.16>, <http://drops.dagstuhl.de/opus/volltexte/2018/9915>
2. Böhm, S., Göller, S.: Language equivalence of deterministic real-time one-counter automata is NL -complete. In: Murlak, F., Sankowski, P. (eds.) Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22–26, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6907, pp. 194–205. Springer (2011). https://doi.org/10.1007/978-3-642-22993-0_20, https://doi.org/10.1007/978-3-642-22993-0_20
3. Böhm, S., Göller, S., Jancar, P.: Equivalence of deterministic one-counter automata is NL -complete. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1–4, 2013. pp. 131–140. ACM (2013). <https://doi.org/10.1145/2488608.2488626>, <https://doi.org/10.1145/2488608.2488626>
4. Boker, U., Kuperberg, D., Kupferman, O., Skrzypczak, M.: Nondeterminism in the presence of a diverse or unknown future. In: Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part II. p. 89–100. ICALP’13, Springer-Verlag, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39212-2_11, https://doi.org/10.1007/978-3-642-39212-2_11
5. Boker, U., Kuperberg, D., Lehtinen, K., Skrzypczak, M.: On the Succinctness of Alternating Parity Good-For-Games Automata. In: Saxena, N., Simon, S. (eds.) 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020). Leibniz International Proceedings in Informatics (LIPIcs), vol. 182, pp. 41:1–41:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2020). <https://doi.org/10.4230/LIPIcs.FSTTCS.2020.41>, <https://drops.dagstuhl.de/opus/volltexte/2020/13282>

6. Boker, U., Kupferman, O., Skrzypczak, M.: How Deterministic are Good-For-Games Automata. In: Lokam, S., Ramanujam, R. (eds.) 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 93, pp. 18:1–18:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). <https://doi.org/10.4230/LIPIcs.FSTTCS.2017.18>, <http://drops.dagstuhl.de/opus/volltexte/2018/8377>
7. Boker, U., Lehtinen, K.: History determinism vs. good for gameness in quantitative automata. In: Bojanczyk, M., Chekuri, C. (eds.) 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15–17, 2021, Virtual Conference. LIPIcs, vol. 213, pp. 38:1–38:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). <https://doi.org/10.4230/LIPIcs.FSTTCS.2021.38>, <https://doi.org/10.4230/LIPIcs.FSTTCS.2021.38>
8. Boker, U., Lehtinen, K.: Token games and history-deterministic quantitative automata. In: Bouyer, P., Schröder, L. (eds.) Foundations of Software Science and Computation Structures. pp. 120–139. Springer International Publishing, Cham (2022)
9. Casares, A., Colcombet, T., Lehtinen, K.: On the size of good-for-games rabin automata and its link with the memory in muller games. In: Bojanczyk, M., Merelli, E., Woodruff, D.P. (eds.) 49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4–8, 2022, Paris, France. LIPIcs, vol. 229, pp. 117:1–117:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). <https://doi.org/10.4230/LIPIcs.ICALP.2022.117>, <https://doi.org/10.4230/LIPIcs.ICALP.2022.117>
10. Colcombet, T.: The theory of stabilisation monoids and regular cost functions. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikolettseas, S., Thomas, W. (eds.) Automata, Languages and Programming. pp. 139–150. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
11. Colcombet, T.: Forms of Determinism for Automata (Invited Talk). In: Dürr, C., Wilke, T. (eds.) 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012). Leibniz International Proceedings in Informatics (LIPIcs), vol. 14, pp. 1–23. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2012). <https://doi.org/10.4230/LIPIcs.STACS.2012.1>, <http://drops.dagstuhl.de/opus/volltexte/2012/3386>
12. Erlich, E., Guha, S., Jecker, I., Lehtinen, K., Zimmermann, M.: History-deterministic parikh automata. CoRR **abs/2209.07745** (2022). <https://doi.org/10.48550/arXiv.2209.07745>, <https://doi.org/10.48550/arXiv.2209.07745>
13. Guha, S., Jecker, I., Lehtinen, K., Zimmermann, M.: A Bit of Nondeterminism Makes Pushdown Automata Expressive and Succinct. In: Bonchi, F., Puglisi, S.J. (eds.) 46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 202, pp. 53:1–53:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). <https://doi.org/10.4230/LIPIcs.MFCS.2021.53>, <https://drops.dagstuhl.de/opus/volltexte/2021/14493>
14. Henzinger, T.A., Lehtinen, K., Totzke, P.: History-deterministic timed automata. In: Klin, B., Lasota, S., Muscholl, A. (eds.) 33rd International Conference on Concurrency Theory, CONCUR 2022, September 12–16, 2022, Warsaw, Poland. LIPIcs, vol. 243, pp. 14:1–14:21. Schloss Dagstuhl - Leibniz-Zentrum für Infor-

- matik (2022). <https://doi.org/10.4230/LIPIcs.CONCUR.2022.14>, <https://doi.org/10.4230/LIPIcs.CONCUR.2022.14>
15. Henzinger, T.A., Piterman, N.: Solving games without determinization. In: Ésik, Z. (ed.) Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4207, pp. 395–410. Springer (2006). https://doi.org/10.1007/11874683_26, https://doi.org/10.1007/11874683_26
 16. Hofman, P., Lasota, S., Mayr, R., Totzke, P.: Simulation problems over one-counter nets. *Log. Methods Comput. Sci.* **12** (2016)
 17. Hofman, P., Mayr, R., Totzke, P.: Decidability of weak simulation on one-counter nets. In: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013. pp. 203–212. IEEE Computer Society (2013). <https://doi.org/10.1109/LICS.2013.26>, <https://doi.org/10.1109/LICS.2013.26>
 18. Hofman, P., Totzke, P.: Trace inclusion for one-counter nets revisited. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8762, pp. 151–162. Springer (2014). https://doi.org/10.1007/978-3-319-11439-2_12, https://doi.org/10.1007/978-3-319-11439-2_12
 19. Holzer, M.: On emptiness and counting for alternating finite automata. In: Dassow, J., Rozenberg, G., Salomaa, A. (eds.) Developments in Language Theory II, At the Crossroads of Mathematics, Computer Science and Biology, Magdeburg, Germany, 17-21 July 1995. pp. 88–97. World Scientific, Singapore (1995)
 20. Hunter, P.: Reachability in succinct one-counter games. In: Bojanczyk, M., Lasota, S., Potapov, I. (eds.) Reachability Problems - 9th International Workshop, RP 2015, Warsaw, Poland, September 21-23, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9328, pp. 37–49. Springer (2015). https://doi.org/10.1007/978-3-319-24537-9_5, https://doi.org/10.1007/978-3-319-24537-9_5
 21. Ibarra, O.H.: Automata with reversal-bounded counters: A survey. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) Descriptive Complexity of Formal Systems - 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8614, pp. 5–22. Springer (2014). https://doi.org/10.1007/978-3-319-09704-6_2, https://doi.org/10.1007/978-3-319-09704-6_2
 22. Jancar, P., Osicka, P., Sawa, Z.: Expspace-hardness of behavioural equivalences of succinct one-counter nets. *CoRR* **abs/1801.01073** (2018), <http://arxiv.org/abs/1801.01073>
 23. Jancar, P., Sawa, Z.: A note on emptiness for alternating finite automata with a one-letter alphabet. *Inf. Process. Lett.* **104**(5), 164–167 (2007). <https://doi.org/10.1016/j.ipl.2007.06.006>, <https://doi.org/10.1016/j.ipl.2007.06.006>
 24. Kucera, A.: On simulation-checking with sequential systems. In: He, J., Sato, M. (eds.) Advances in Computing Science - ASIAN 2000, 6th Asian Computing Science Conference, Penang, Malaysia, November 25-27, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1961, pp. 133–148. Springer (2000). https://doi.org/10.1007/3-540-44464-5_11, https://doi.org/10.1007/3-540-44464-5_11
 25. Kuperberg, D., Skrzypczak, M.: On determinisation of good-for-games automata. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) Automata, Languages, and Programming - 42nd International Colloquium, ICALP

- 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9135, pp. 299–310. Springer (2015). https://doi.org/10.1007/978-3-662-47666-6_24, https://doi.org/10.1007/978-3-662-47666-6_24
26. Lehtinen, K., Zimmermann, M.: Good-for-games ω -pushdown automata. *Log. Methods Comput. Sci.* **18**(1) (2022). [https://doi.org/10.46298/lmcs-18\(1:3\)2022](https://doi.org/10.46298/lmcs-18(1:3)2022), [https://doi.org/10.46298/lmcs-18\(1:3\)2022](https://doi.org/10.46298/lmcs-18(1:3)2022)
 27. Martin, D.A.: Borel determinacy. *Annals of Mathematics* **102**(2), 363–371 (1975), <http://www.jstor.org/stable/1971035>
 28. Radi, B.A., Kupferman, O.: Minimization and canonization of GFG transition-based automata. *Log. Methods Comput. Sci.* **18**(3) (2022). [https://doi.org/10.46298/lmcs-18\(3:16\)2022](https://doi.org/10.46298/lmcs-18(3:16)2022), [https://doi.org/10.46298/lmcs-18\(3:16\)2022](https://doi.org/10.46298/lmcs-18(3:16)2022)
 29. Srba, J.: Visibly pushdown automata: From language equivalence to simulation and bisimulation. In: Ésik, Z. (ed.) *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings. Lecture Notes in Computer Science*, vol. 4207, pp. 89–103. Springer (2006). https://doi.org/10.1007/11874683_6, https://doi.org/10.1007/11874683_6
 30. Valiant, L.G.: Decision procedures for families of deterministic pushdown automata. Ph.D. thesis, University of Warwick, Coventry, UK (1973), <http://wrap.warwick.ac.uk/34701/>
 31. Valiant, L.G., Paterson, M.S.: Deterministic one-counter automata. *Journal of Computer and System Sciences* **10**(3), 340–350 (1975). [https://doi.org/https://doi.org/10.1016/S0022-0000\(75\)80005-5](https://doi.org/https://doi.org/10.1016/S0022-0000(75)80005-5), <https://www.sciencedirect.com/science/article/pii/S0022000075800055>

A Appendix for Section 3

A.1 Simulation Games

We argue that deciding the winner in simulation game (cf. definition 9) is logspace interreducible to deciding the winner in simulation games when the winning condition is given solely by the inability of the either players to choose transitions, and not by accepting states.

Note that in Definition 9 for simulation games, we can complete both the one-counter nets by adding a rejecting sink state in each of them, from which we have a transition from every state on Σ that does not change the counter value. We also add self loops on Σ on the sink state that do not change the counter value. This slight modification does not change the winner in the simulation game.

Consider the following decision problem, which we call SIMULATION:

Given: Two complete one-counter nets \mathcal{N} and \mathcal{M} , and configurations $(p, k) \in \mathcal{C}(\mathcal{N})$ and $(p', k') \in \mathcal{C}(\mathcal{M})$
Question: Does \exists win the simulation game $\mathcal{G}(\mathcal{N}, (p, k) \longleftrightarrow \mathcal{M}, (p', k'))$?

We formally define the OriginalSim game, which is the simulation game where the winning condition is given by the inability of the either player to choose transitions, as defined in literature [16].

Definition 29. Let $\mathcal{N} = (Q, \Sigma, q_0)$ and $\mathcal{M} = (Q, \Sigma', \Delta')$ be two one-counter nets. Given two configuration (p, k) and (p', k') in \mathcal{N} and \mathcal{M} respectively with $k, k' \in \mathbb{N}$, the OriginalSim game between \mathcal{N} and \mathcal{M} at position $((p, k), (p', k'))$, is a two player game between \forall and \exists , with positions in $\mathcal{C}(\mathcal{N}) \times \mathcal{C}(\mathcal{M})$ where the initial position is $((p_0, k_0), (p'_0, k'_0)) = ((p, k), (p', k'))$. At round i of the play, where the position is $((p_i, k_i), (p'_i, k'_i))$:

- \forall selects a letter $a \in \Sigma$, and a transition $(p_i, k_i) \xrightarrow{a, d} (p_{i+1}, k_{i+1})$ in \mathcal{N}
- \exists selects an a -transition $(p'_i, k'_i) \xrightarrow{a, d'} (p'_{i+1}, k'_{i+1})$ in \mathcal{M}

If after choosing a letter, \forall can't choose a transition, then \forall loses. If after \forall having chosen a transiting, \exists is unable to choose a transition, then \exists loses. Else, the game goes to $((p_{i+1}, k_{i+1}), (p'_{i+1}, k'_{i+1}))$ for another round of the play. The player \exists wins any infinite play.

We will also call ORIGINALSIM, the decision problem of asking if \exists wins the game defined above.

Given: Two one-counter nets \mathcal{N} and \mathcal{M} , and configurations $(p, k) \in \mathcal{C}(\mathcal{N})$, and $(p', k') \in \mathcal{M}$,
Question: Does \exists win the OriginalSim game between \mathcal{N} and \mathcal{M} at position $((p, k), (p', k'))$?

We now show that the two problems are log-space inter-reducible to each other for asking the decision problem about the winner of the game. A closer examination of the proofs will also reveal a more nuanced equivalence between the two definitions.

Reducing SIMULATION to ORIGINALSIM: Given an instance of problem SIMULATION, with two complete one-counter nets $\mathcal{N} = (Q, \Sigma, \Delta, q_0, F)$ and $\mathcal{M} = (Q', \Sigma, \Delta', q'_0, F')$, and configurations $(p, k) \in \mathcal{C}(\mathcal{N})$ and $(p', k') \in \mathcal{C}(\mathcal{M})$, we reduce it to an instance of ORIGINALSIM. We construct the (not necessarily complete) net $\mathcal{N}' (\mathcal{M}')$ by introducing a new alphabet $\$ \notin \Sigma$ to $\mathcal{N} (\mathcal{M})$, and adding self loops on $\$$ on final states in $\mathcal{N} (\mathcal{M})$ that do not change the counter. That is, for each state $q \in F$ ($q \in F'$), we introduce the transition $(q, \$, 0, q)$.

We claim that for any configurations $((p, k), (p', k')) \in \mathcal{C}(\mathcal{N}) \times \mathcal{C}(\mathcal{M})$, the player \exists wins the simulation game $\mathcal{G}(\mathcal{C}(\mathcal{N}), (p, k) \hookrightarrow \mathcal{C}(\mathcal{M}), (p', k'))$ if and only if the player \exists wins the OriginalSim game between \mathcal{N} and \mathcal{M} at position $((p, k), (p', k'))$.

\Rightarrow : Suppose \exists plays in the OriginalSim game between \mathcal{N}' and \mathcal{M}' according to a winning positional strategy in the simulation game between \mathcal{N} and \mathcal{M} whenever \forall chooses a letter in Σ . Note that there is at most one $\$$ transition on each state, so \exists either has no choice or a unique choice for choosing a transition in the simulation game at her token.

If \forall never chooses $\$$ in OriginalSim game between \mathcal{N}' and \mathcal{M}' , then \exists wins the game as both \mathcal{N}' and \mathcal{M}' are complete on Σ . Now, whenever \forall chooses $\$$ and a transition on $\$$ in \mathcal{N}' , then \forall 's token in \mathcal{N}' must have been at a state which is accepting in \mathcal{N} . As \exists was playing according to her winning strategy, \exists 's token in \mathcal{M}' would have been at a state corresponding to an accepting one in \mathcal{M} , which means \exists would be able to take $\$$ as well. Note that taking a $\$$ transition does not change the counter value in both \mathcal{M}' and \mathcal{N}' . Thus, \exists is able to choose a transition whenever \forall can choose one, and hence \exists wins OriginalSim game.

\Leftarrow : Suppose \exists wins the OriginalSim game between \mathcal{N}' and \mathcal{M}' , and \exists plays in the simulation game according to a winning strategy in the OriginalSim game. Then, whenever \forall 's token in \mathcal{N} is at a final state in the simulation game, \exists 's must be at a final state as well. If not, then \forall would have been able to take a $\$$ transition in the OriginalSim game, while \exists wouldn't be able to, which contradicts the fact that \exists was playing according to a winning strategy,

Reducing ORIGINALSIM to SIMULATION: Given an instance of problem ORIGINALSIM, with two one-counter nets $\mathcal{N}' = (Q, \Sigma, \Delta, q_0)$ and $\mathcal{M}' = (Q', \Sigma, \Delta', q'_0)$, and configurations $(p, k) \in \mathcal{C}(\mathcal{N})$ and $(p', k') \in \mathcal{C}(\mathcal{M})$, we reduce it to an instance of SIMULATION. We construct $\mathcal{N} (\mathcal{M})$ by completing the net by adding transitions on Σ that do not change the counter value to a sink state s (s') which is rejecting, and making all the original states Q (Q') accepting.

We claim that for any configuration $((p, k), (p', k')) \in \mathcal{C}(\mathcal{N}') \times \mathcal{C}(\mathcal{M}')$, then \exists wins the OriginalSim game between \mathcal{N}' and \mathcal{M}' at position $((p, k), (p', k'))$ if and only if the player \exists wins the simulation game $\mathcal{G}(\mathcal{C}(\mathcal{N}), (p, k) \hookrightarrow \mathcal{C}(\mathcal{M}), (p', k'))$.

\Rightarrow : Suppose \exists plays in the simulation game between \mathcal{N} and \mathcal{M} according to a winning strategy in the OriginalSim game between \mathcal{N}' and \mathcal{M}' . Note that once \forall 's token goes to the sink state s which is rejecting, then \exists wins the simulation game, as \forall will never see an accepting state. If \forall stays in the states of \mathcal{N}' that were also in \mathcal{N} , then \exists would also be able to stay in the states of \mathcal{M}' that were

also in \mathcal{M} . As these states comprises of all accepting states in \mathcal{N} and \mathcal{M} , this implies that whenever \forall 's token in \mathcal{N} is at an accepting state, so is \exists 's.

\Leftarrow : Suppose \exists wins the simulation game between \mathcal{N} and \mathcal{M} , and \exists plays in the OriginalSim game according to a winning strategy for \exists in the simulation game. Then, at any position of a play according to \exists 's strategy, whenever \forall is able to take a transition on a in \mathcal{N}' , then \exists must be able to take a transition in \mathcal{M}' according to her strategy as well : If not, then in the simulation game, \exists 's token would be at a rejecting state in \mathcal{M} while \forall 's would be at an accepting state in \mathcal{N} , which contradicts the fact that \exists was playing according to a winning strategy in the simulation game.

A.2 Proof of Lemma 6

To prove the above lemma, we need to understand better the structure of the resolvers for OCNs. Consider the definition given below of *residual transitions*. Intuitively, these are transitions such that if there was an accepting run from a state with the first letter as a , on taking a residual transition on a , then there is still an extension of the run from the new state. More formally, we say that a transition $(q, k) \xrightarrow{a, d} (q', k')$ is *residual* if $L(q', k') = a^{-1}L(q, k)$, where $L(q, k)$ (and $L(q', k')$) is the set of words that are accepted in \mathcal{N} when the initial configuration is (q, k) ((q', k')), instead of $(q_0, 0)$. We claim that any winning strategy of \exists can be characterised by these residual transitions.

We recall from the paper, stated again

Proposition 30 (also, Proposition 7). *For an OCN \mathcal{N} , a strategy σ for \exists in the letter game is winning for \exists if and only if σ takes only residual transitions.*

Proof. \Rightarrow : Suppose, $\sigma(q, k, u, a) = (q', k) \xrightarrow{a, d} (q', k')$, for some configuration (q, k) , reached upon reading a prefix u following σ . We need to show that $L(q', k') = a^{-1}L(q, k)$. But if there is a word $w \in a^{-1}L(\mathcal{N}, (q, k)) \setminus L(\mathcal{N}, (q', k'))$, then on the word uaw , the strategy σ can't end at an accepting state, but $uaw \in L$, a contradiction.

\Leftarrow : Suppose σ is an \exists strategy which only takes residual transitions. Then for each word w , if (q, k) is the configuration reached upon reading the word w , then $L(\mathcal{N}, (q, k)) = w^{-1}L$. If $w \in L$, then $\epsilon \in w^{-1}L$, and hence the configuration (q, k) is accepting. Thus, σ is at an accepting state whenever the word read so far is accepting, and thus σ is a winning strategy.

Note that in the letter game, each player winning the game has a positional winning strategy, as it is a reachability game. Suppose \exists wins the letter game, then \exists has a winning strategy which can be given by a (partial) function

$$\sigma : (Q \times \mathbb{N}) \times \Sigma^* \times \Sigma \rightarrow \Delta^*.$$

Using Proposition 7, we can show that \exists 's strategy only depends on the configuration, and is independent of the word read so far. Proposition 8 below show that we can have a resolver based on the current state and counter value alone.

Proposition 31 (also, Proposition 8). *If \exists wins the letter game on an OCN \mathcal{N} , then \exists has a winning strategy σ that only depends on the current configuration of the play, i.e σ is a partial function $\sigma : (Q \times \mathbb{N}) \times \Sigma \rightarrow \Delta^*$*

Proof. Let $\sigma' : (Q \times \mathbb{N}) \times \Sigma^* \times \Sigma \rightarrow \Delta^*$ be any winning strategy. We define a strategy σ as follows: For each configuration (q, k) in the one-counter net \mathcal{N} , we let $\sigma((q, k), a) = (q, k) \xrightarrow{a, d} (q', k')$, where for some word u on which σ' reaches the configuration (q, k) , we have $\sigma'((q, k), u, a) = (q, k) \xrightarrow{a, d} (q', k')$. By the Proposition 7, σ' takes only residual transitions, implying σ takes only residual transitions as well, and it follows from proposition 7 that σ is a winning strategy for \exists which depends only on the configuration.

Having characterised what strategies look like in the letter game, we are finally equipped to prove Lemma 6 using Proposition 7 and Proposition 8.

Finally, we prove Lemma 6 Let γ be a winning strategy for \exists in G_1 , and let λ be the strategy in the letter game, derived from γ where \forall copies \exists 's play. We show that λ takes only residual transitions. This is enough because of Proposition 7. Assume to the contrary, that \exists takes a non-residual transition in a play following γ , $(p, k) \xrightarrow{a, d} (p', k')$ at some position in G_1 . As \forall 's token is also at (p, k) , \forall can win by constructing a word $av \in \mathcal{L}(p, k)$, such that $v \notin \mathcal{L}(p', k')$ so that \exists can't produce an accepting run of v from (p', k') , while constructing an accepting run for $a \cdot v$ from (p, k) with his token. Thus, \exists loses G_1 , and γ is not a winning strategy in G_1 , a contradiction.

A.3 Proof of Lemma 11

We state the construction's intuition as well as give a rigorous construction of the nets \mathcal{M} and \mathcal{M}' side-by-side for ease of reference. Let $\mathcal{N} = (Q, \Sigma, \Delta, q_0, F)$.

Construction of \mathcal{M}' We define \mathcal{M} to essentially be the net \mathcal{N} , but containing an extra state $q_\#$, and a newly added letter $\#$. Along with this, there are new transitions added. These transitions loop on the new state for any letter in the expanded alphabet. Transitions are added to reach this state $q_\#$ by reading $\#$ from a final state in \mathcal{N} . Formally, $\mathcal{M}' = (Q', \Sigma_\#, \Delta', q_0, F')$, where

- the set of states is $Q' = Q \cup \{q_\#\}$, $\Sigma_\# = \Sigma \cup \{\#\}$,
- the accepting state is just $F' = \{q_\#\}$, and
- the set of transitions Δ' is defined as

$$\Delta' = \Delta \cup \{(q, \#, 0, q_\#) | q \in F\} \cup \{(q_\#, a, 0, q_\#) | a \in \Sigma_\#\}.$$

Note that \mathcal{M}' has the same initial state q_0 , as \mathcal{N} .

Construction of \mathcal{M} We construct \mathcal{M} to contain approximately $(|\Sigma| + 1)$ copies of the states in \mathcal{N} . These copies help remember the previous letter read in the state space of the OCN. The transitions are such that, on reading a letter, the letter is ‘stored’ in the state space. However, in the projection of the \mathcal{N} part of \mathcal{M} ’s state, the new state of \mathcal{N} that the transition takes it to, is based on the letter that was previously stored in the letter component, as opposed to the current letter read, which will now be stored in the current state space. This is built to capture a play of G_1 by creating a delay in the simulation game for \forall by forcing a one-step delay during his play. This turns out to be important to capture the G_1 game. We formalise the above intuition below by defining $\mathcal{M} = (Q_M, \Sigma_\#, \Delta_M, s, F_M)$, where

- $Q_M = (Q \times \Sigma) \cup \{s\} \cup (F \times \{\#\})$,
- s is the initial state,
- F_M , the set of final states is $F \times \{\#\}$, and
- the set of transitions Δ_M is the union of the following sets :
 - $\{(s, a, 0, (q_I, a)) \mid a \in \Sigma\}$
 - $\{((p, a), b, d, (q, b)) \mid (p, a, d, q) \in \Delta\}$
 - $\{((p, a), \#, d, (q, \#)) \mid (p, a, d, q) \in \Delta \text{ and } q \in F\}$

Winning in G_1 implies winning in $\mathcal{G}(\mathcal{M} \hookrightarrow \mathcal{M}')$ Suppose \exists wins G_1 in \mathcal{N} . Then, in the simulation game $\mathcal{G}(\mathcal{M} \hookrightarrow \mathcal{M}')$, \exists can win by inductively constructing a simultaneous play of G_1 in her memory:

- The simulation game starts at $(s, 0), (q_I, 0)$. For any letter $a \in \Sigma$ that \forall picks (if he picks $\#$, he loses, as he can’t move from $\#$ on s), the transitions available take him to (q_I, a) , since this transition is deterministic. To respond to the above play, \exists uses the winning strategy of G_1 . In \exists ’s view she would respond with the transition that she would have in G_1 if \forall picked $a \in \Sigma$. Suppose the strategy in G_1 for \exists was the transition $(q_I, 0) \xrightarrow{a, k_1} (q_1, k_1)$, then she uses this as her strategy to produce a transition in \mathcal{M}' in the simulation game. She also builds a G_1 play where she has made a move from the initial state to (q_1, k_1) and where \forall is yet to make a move in the corresponding G_1 game. Therefore their configurations in G_1 are at $((q_1, k_1), (q_I, 0))$, waiting for \forall to move from q_I .
- Suppose, after i rounds where \forall has not having played $\#$ until then. Let the simulation game be at the position $((p_{i-1}, a), k), (q_{i-1}, k')$. Suppose inductively, the corresponding run of \exists in G_1 for such a play in \mathcal{N} is $((q_i, k'_i), (p_{i-1}, k_{i-1}))$ where \forall is yet to make a move on a from the configuration (p_{i-1}, k_{i-1}) . In the simulation game $\mathcal{G}(\mathcal{M} \hookrightarrow \mathcal{M}')$,
 - \forall chooses a letter b , and a corresponding transition $((p_{i-1}, a), b, d, (p_i, b))$ from the available transitions of \mathcal{M} .
 - \exists responds as though, the transition $(p_{i-1}, k_{i-1}) \xrightarrow{a, d} (p_i, k_i) \in \Delta$, was played by \forall as move continuing G_1 from the tuple of configurations $((q_i, k'_i), (p_{i-1}, k_{i-1}))$ to the tuple of configurations $((q_i, k'_i), (p_i, k_i))$. Since \exists has a strategy in G_1 , this strategy prescribes a transition, say $(q_i, k'_i) \xrightarrow{b, d'} (q_{i+1}, k'_{i+1})$ where \forall has picked the letter b in the G_1 game.

She picks the same transition available to her in \mathcal{M}' as a response in the simulation game.

After the transition $(q_i, k'_i) \xrightarrow{b, d'} (q_{i+1}, k'_{i+1})$ was picked, the corresponding inductive game of G_1 built is updated to the tuple of configurations $(q_{i+1}, k'_{i+1}), (p_i, k_i)$ where \forall needs to pick a transition on b where, \forall 's run constructed in the G_1 game is at (p_i, k_i) .

- Suppose \forall picks $\#$ at some position i , and suppose the configuration of the tokens were at $((p_{i-1}, a), k_{i-1}), (q_i, k'_i)$, then \forall can make a move if and only if \forall can get to a final state, by a transition (p_{i-1}, a, d, p_{i+1}) , for $p_{i+1} \in F$. But then, as \exists is winning on G_1 , she must have constructed an accepting run and hence \exists must have been at a final state as well. This enables \exists to move her token would to $q_\#$ in the simulation game from which she can win.

Thus, we have shown that if \exists wins G_1 then \exists wins the simulation game $\mathcal{G}(\mathcal{M} \hookrightarrow \mathcal{M}')$ as well.

Winning in $\mathcal{G}(\mathcal{M} \hookrightarrow \mathcal{M}')$ implies winning in G_1 We will now show the other direction that if \exists wins the simulation game $\mathcal{G}(\mathcal{M} \hookrightarrow \mathcal{M}')$, then \exists can also win G_1 . Since \exists has a winning strategy in simulation game $\mathcal{G}(\mathcal{M} \hookrightarrow \mathcal{M}')$, we will show how to modify it to obtain a winning strategy for \exists in G_1 . For any play in the simulation game, we will inductively build a play in G_1 and show that a strategy used in the simulation game also produces a corresponding winning strategy in G_1 . However, we consider a slightly different linearisation of the game G_1 . We make \forall to wait one more turn to execute his 3^{rd} component of the play. For the first position where both the tokens are at the initial configuration, we say

1. \forall picks a letter $a \in \Sigma$.
2. \exists responds by picking a transition $(q_I, 0) \xrightarrow{a, k_1} (q_1, k_1)$.

Now, we say that the token is at position (q_1, k'_1) for \exists and $(q_I, 0)$ with \forall having to execute a . For the i^{th} turn, from configurations (q_i, k'_i) for \exists and (p_{i-1}, k_{i-1}) for \forall with him having to execute a letter $a_{i-1} \in \Sigma$,

1. \forall picks a transition $(p_{i-1}, k_{i-1}) \xrightarrow{a_{i-1}, d} (p_i, k_i)$ as well as a letter $a_i \in \Sigma$.
2. \exists responds by picking a transition $(q_i, k'_i) \xrightarrow{a_i, d'} (q_{i+1}, k'_{i+1})$ on a_i .

The winning condition for \exists is the following: If at some round i , the player \forall is at a configuration (p_{i-1}, k_{i-1}) such that \forall can pick a transition $(p_{i-1}, k_{i-1}) \xrightarrow{a_{i-1}, d} (p_i, k_i)$ with $p_i \in F$, then it must be that q_i was already a final state. The above modified formalisation of G_1 makes it easier to see the equivalence with the simulation game of the two automata constructed. The proof proceeds inductively.

- The newly defined modification of the game G_1 starts while all the tokens for each of the player both correspond to the configurations $(q_I, 0), (q_I, 0)$. In G_1 ,

- For any letter $a \in \Sigma$ picked by \forall , we consider the corresponding play of \forall in the game $\mathcal{G}(\mathcal{M} \longleftrightarrow \mathcal{M}')$ here:
 - * \forall picks a transition from $(s, 0) \xrightarrow{a, 0} ((q_I, a), 0)$. In the simulation game,
 - * \exists responds with a transition of \mathcal{M}' suggested by her winning strategy, say with the transition $(q_I, 0) \xrightarrow{a, k'_1} (q_1, k'_1)$.
 - For the game G_1 , player \exists is to respond also with the corresponding transition $(q_I, 0) \xrightarrow{a, k'_1} (q_1, k'_1)$ above, but in the net \mathcal{N} during her turn. So the configuration in the game G_1 is where \exists is at (q_1, k'_1) and \forall at $(q_I, 0)$, waiting to make a move on a .
 - In the i^{th} turn of the modified game G_1 , we assume that the game G_1 is at configuration (q_i, k'_i) for \exists and (p_{i-1}, k_{i-1}) for \forall with him having to execute a letter $a_{i-1} \in \Sigma$. The inductive run of the simulation game must therefore be at the following position: (q_i, k'_i) in the net \mathcal{M}' and $((p_{i-1}, a_{i-1}), k_{i-1})$ in the net \mathcal{M} for this round of the simulation game:
 - \forall picks a transition over \mathcal{M} , say $((p_{i-1}, a_{i-1}), k_{i-1}) \xrightarrow{a_i, d} ((p_i, a_i), k_i)$. This corresponds to a unique transition over a_{i-1} along with another letter $a_i \in \Sigma$ in the net \mathcal{N} , by the construction of \mathcal{M} . We assume that \forall in G_1 extends the run with the above pair of letter and transition
 - Since \exists has a winning strategy, she extends the run accordingly, by responding with a transition $(q_i, k'_i) \xrightarrow{a_i, d'} (q_{i+1}, k'_{i+1})$ from the net \mathcal{M}'
- The same transition is picked by \exists in G_1 : $(q_i, k'_i) \xrightarrow{a_i, d'} (q_{i+1}, k'_{i+1})$. This transition is available in \mathcal{N} by construction. Therefore, the game G_1 is now at configurations: (q_{i+1}, k'_{i+1}) for \exists and (p_i, k_i) for \forall , who is waiting to make a move on a_i . The corresponding simulation game is at the configuration $((p_i, a_i), k_i)$ for \forall and (q_{i+1}, k'_{i+1}) for \exists
- In the i^{th} turn of the modified G_1 game, we argue that if at the configuration (q_i, k'_i) for \exists and (p_{i-1}, k_{i-1}) for \forall with him having to execute a letter $a_{i-1} \in \Sigma$, if after executing an a_{i+1} transition on $(p, i-1, k_{i-1})$ he can end up in an accepting state, then \exists can also win G_1 , as she should already be in an accepting state. This follows from the fact that the simulation game is winning. Especially because the run constructed so far was based on a winning strategy of eve in the simulation game. This inductively built run would be at a configuration (q_i, k'_i) in the net \mathcal{M}' and $((p_{i-1}, a_{i-1}), k_{i-1})$ in the net \mathcal{M} , before \forall moves. Consider the following sequence of moves in the simulation game:
 - \forall picks a configuration, and a letter, and if the word read so far is accepting, then he could pick a transition that leads to a final state along with $\#$.
 - \exists would have to respond with a $\#$ -transition but that is only available from states q_i , such that q_i is a final state, showing that \exists can win henceforth, because she moves in \mathcal{M}' to a state where all words are accepting.

A.4 Proof of Lemma 14

Proof. The proof goes by reducing from the problem of checking non-emptiness of alternating finite-state automata over a unary alphabet. This variation of the problem was proven to be **PSPACE**-complete by Holzer [19], with its proof simplified by Jančar and Sawa [23].

We define an alternating finite-state automata over the unary alphabet as follows: $\mathcal{A} = (Q = Q_\vee \uplus Q_\wedge, \delta, q_0, F)$ where Q is a finite set of states, partitioned among two players \vee and \wedge , $q_0 \in Q$ is the start state, $F \subseteq Q$ is the final state and the transitions are $\delta \subseteq (Q_\vee \times Q_\wedge) \cup (Q_\wedge \times Q_\vee)$.

The empty-string ϵ , which has length 0 is in the language accepted by \mathcal{A} from $q \in Q$ iff $q \in F$. We say a word of length $n \geq 1$ is accepted from q if either

- $q \in Q_\vee$ and there exists $(q, q') \in \delta$ such that a word of length $n - 1$ accepted by \mathcal{A} from q' ; or
- $q \in Q_\wedge$ and for every $(q, q') \in \delta$, there is a word of length $n - 1$ accepted by \mathcal{A} from q' .

A word of length n is accepted by \mathcal{A} if it is accepted from the initial state q_0 .

Intuitively, this can be thought of as a game between two players, \vee and \wedge where an n length word is accepting in the automaton if and only if \vee wins in the n -length play in the above automaton viewed as a reachability game to one of the final states. The player \wedge 's objective is adversarial to \vee .

Given such an alternating finite automaton, we construct a one-counter net \mathcal{H} that is history-deterministic if and only if \mathcal{A} is empty.

We would like to show that

- \forall wins the letter game on \mathcal{H} if there is a word accepted by \mathcal{A}
- \exists wins the letter game on \mathcal{H} when \mathcal{A} is empty

The idea is that we add a state q_I , which will be a new initial state of the one-counter net. On q_I , \forall can read a special input **1**, which will increase the counter value while reading this input. To win, \forall would have to read as many **1**s as the length of an accepting word, and later prove that indeed this word is accepting. While constructing the run, since \forall is the one proving non-emptiness, he will be resolving what we would think of as ‘existential’ choices, here denoted by transitions of \vee . The player \exists on other hand, would be resolving the ‘universal’ choices which are the transitions of \wedge .

A run is constructed by having a copy of states and encoding in the alphabet, the choices to be made by \vee player so that \forall can pick the letter. For the \wedge player, we want \exists to make the ‘universal’ choice, so we encode this in the non-determinism. But to ensure \forall ensures eve to ‘fairly’ pick the choices, if \forall decides to read a word state that does not respect the current state that \exists is in, she can move to a state from which there is no non-determinism resolution, and \exists can accept any valid suffix.

If \forall has reached a final state with counter value exactly 0, then he has displayed that such an accepting run exists, from where \forall reads a \$, and can win the letter game. But if this run has reached a final state with a positive

value, and \forall reads a \$, then \exists can win the letter game. This is done by a gadget described as follows:

From any state, \forall can read a symbol \$. If the counter is non-zero, or if the state is non-accepting, \exists has a transition that subtracts 1 from the counter and goes to a state from which any of the two special symbols are accepted: \heartsuit and \clubsuit . Whereas, if the same symbol \$ is read by \forall when the counter is empty and at an accepting state, then the only transitions enabled have a non-determinism that cannot be resolved by \exists , where she would have to predict if \heartsuit or \clubsuit will be seen in the future. We refer the reader to a pictorial representation of the

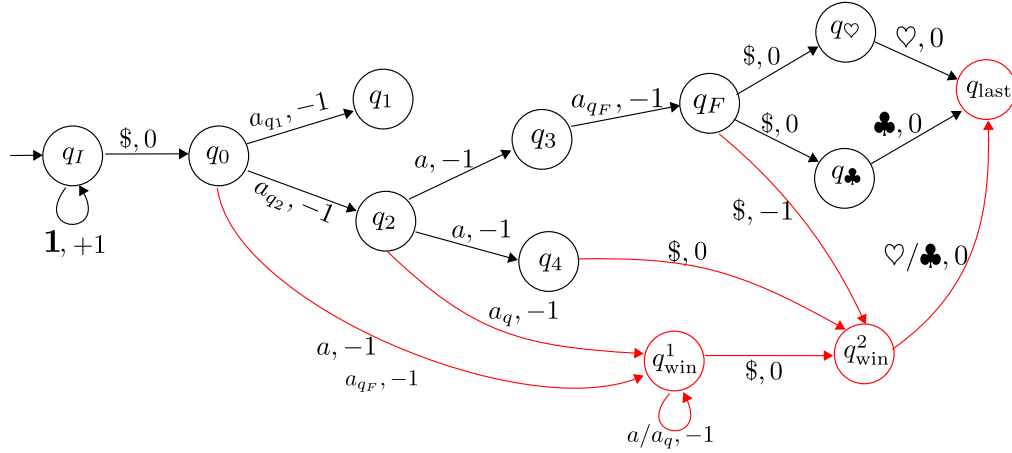


Fig. 4. Some of the states in the net \mathcal{H} constructed. Here, we assume that $q_0, q_3, q_4 \in Q_{\wedge}$ whereas $q_1, q_2, q_F \in Q_{\vee}$. All states are accepting. Observe that at q_F , a final state, the non-determinism is history-deterministic only when the counter is non-empty. Also notice that the red arrows/states are when \forall proposes a letter that does not correspond to a feasible transition played by \exists in the AFA word resolution.

construction in Figure 4.

We define the OCN $\mathcal{H} = (Q_H, \Sigma, \Delta_H, q_I, F_H)$ where

- $Q_H = Q \cup \{q_I\} \cup \{q_{\clubsuit}, q_{\heartsuit}\} \cup \{q_{\text{win}}^1, q_{\text{win}}^2\} \cup \{q_{\text{last}}\}$,
- $\Sigma = \{\$, \heartsuit, \clubsuit, \mathbf{1}\} \cup \{a_q \mid q \in Q_{\vee}\} \cup \{a\}$,
- $F_H = Q_H$, where all states defined are final states, and
- Δ_H , the transitions are the union of the sets of transition below
 1. $\{(q_I, \mathbf{1}, 1, q_I), (q_I, \$, 0, q_0)\}$
 2. $\{(q_{\vee}, a_q, -1, q) \mid (q_{\vee}, q) \in \Delta \text{ and } q_{\vee} \in Q_{\vee}\}$
 3. $\{(q_{\vee}, a_q, -1, q_{\text{win}}^1) \mid (q_{\vee}, q) \notin \Delta \text{ and } q_{\vee} \in Q_{\vee}\}$
 4. $\{(q_{\vee}, a, -1, q_{\text{win}}^1) \mid q_{\vee} \in Q_{\vee}\}$
 5. $\{(q_{\wedge}, a, -1, q) \mid (q_{\wedge}, q) \in \Delta \text{ and } q_{\wedge} \in Q_{\wedge}\}$
 6. $\{(q_{\wedge}, a_p, -1, q_{\text{win}}^1) \mid p \in Q \text{ and } q_{\wedge} \in Q_{\wedge}\}$
 7. $\{(q_F, \$, -1, q_{\text{win}}^2), (q_F, \$, 0, q_{\clubsuit}), (q_F, \$, 0, q_{\heartsuit}) \mid q_F \in F\}$

8. $\{(q, \$, 0, q_{\text{win}}^2) \mid q \notin F\}$
9. $\{(q_{\text{win}}^1, \$, 0, q_{\text{win}}^2)\}$
10. $\{(q_{\text{win}}^1, b, -1, q_{\text{win}}^1) \mid b = a_q \text{ or } b = a\}$
11. $\{(q_{\heartsuit}, \heartsuit, 0, q_{\text{last}}), (q_{\clubsuit}, \clubsuit, 0, q_{\text{last}}), (q_{\text{win}}^2, \heartsuit, 0, q_{\text{last}}), (q_{\text{win}}^2, \clubsuit, 0, q_{\text{last}})\}$.

The state space consists of a state q_I , and a copy of the states Q of \mathcal{A} . Moreover, there are states q_{win}^1 and q_{win}^2 from which intuitively, all words that are in the language of \mathcal{H} henceforth are accepted. These states will be winning for \exists if she reaches them in the letter game, and hence \forall loses from that state. There are also two states q_{\heartsuit} and q_{\clubsuit} which ideally \forall would like \exists to reach in the letter game. Intuitively, if \exists reaches a copy of Q 's final states with 0 in the counter, she loses as she has to pick between the states q_{\heartsuit} and q_{\clubsuit} and this kind of non-determinism makes it winning for \forall .

At state q_I , \forall can read as many 1s as he wants, which increases his counter value. Alternatively, there is a symbol $\$$ that he can read when the play moves from q_I to q_0 .

This signals that \forall is ready to display that he can construct a branch of the run-tree with length exactly the counter value. Once he has finished this run construction, he can again use $\$$ to signal the end of a run-constructed at the states, producing the non-determinism that makes \exists win iff the counter is non-zero, or if the state is not accepting.

The letters a_q enables transitions from states which belong to \vee , whereas a enables arbitrary non-deterministic choice consistent with the original automaton's transitions from states belonging to \wedge . From q , the letter a_q enables transitions that are deterministic.

All states in the net \mathcal{H} are final, but the net \mathcal{H} is not complete, making it non-universal.

The transitions are mostly as explained before, but we supply some additional discussion to understand better. At the initial state, \forall could read a counter value and increase arbitrarily. After this he can read $\$$ eventually and enter q_0 . On entering q_0 , the initial state in \mathcal{A} , \forall resolves choices of the player \vee which can be thought of as 'existential' choices whereas \exists the 'universal' choices, the choices of \wedge . This is done by encoding this in the alphabet and non-determinism respectively. While each choice is made, the counter value is decreased as a count-down to the length of the word.

We would like to emphasise here that we add transitions to state q_{win}^1 if \forall reads a letter that does not extend the transitions picked by \exists 's run constructed. This enables \exists to pick her non-deterministic transition in such a way that \forall cannot ensure her loss trivially. This is done by adding transitions from every state in the copy of Q , to a state q_{win}^1 for letters that are such that they do not extend a transition picked by eve while resolving non-determinism. This is made more precise in the definitions of the transitions. If he does pick a correct run then \exists can also only construct a run, and cannot reach q_{win}^1 .

There are several transitions on $\$$. The idea here is that \forall can read $\$$ once he is at a final state. Note that he can also read $\$$ before, but then the play goes to another state q_{win}^2 , winning for \exists from all states that are not final. Similarly, if he

does read \$ from a final state and the counter value is not zero, transitions are enabled for \exists that reach q_{win}^2 . Finally, if the counter value is indeed zero, then the only two transitions enabled make \exists pick in advance for going to state q_{\heartsuit} and q_{\clubsuit} . From these two states there is only one transition \heartsuit and \clubsuit respectively.

At q_{win}^1 however, \exists can reach accepting state on any series of letters a or a_q , whilst decrementing the counter, then seeing a \$ and then read \heartsuit or \clubsuit with no non-determinism.

Note that the language accepted by \mathcal{H} is the prefix closure of

$$\{1^n \$ \cdot a_1 \cdot a_2 \cdot \dots \cdot a_k \cdot \$ \cdot \{\heartsuit, \clubsuit\} \mid a_i = a \text{ or } a_i = a_q \text{ for some } q \in Q, k \leq n\}$$

Proof of correctness of the construction We now proceed to showing that the constructed automaton indeed satisfies the following:

- \Rightarrow \forall wins the letter game on \mathcal{H} if there is a word accepted by \mathcal{A}
- \Leftarrow \exists wins the letter game only when \mathcal{A} is empty

\forall wins the letter game on \mathcal{H} if \mathcal{A} is non-empty We give a strategy for \forall in the letter game: If \mathcal{A} is non-empty, there is some n for which there is an n -length word that is accepting by \mathcal{A} . Without loss of generality, we assume that $n > 0$.

\forall reads the letter 1 n -many times. There is no non-determinism for \exists to resolve in this game so far. After this, \forall reads \$ and the game moves to q_0 , the copy of the initial state of \mathcal{A} .

On reaching the copies of the states of \mathcal{A} , the letter game proceeds following the invariant

the run constructed so far by \exists in the letter game is at a state q , such that there is a word accepted by \mathcal{A} of length equal to the current counter value from state q .

This is indeed true at the vertex q_0 , by assumption that there exists a word of length n accepted from \mathcal{A} , and the counter value has n in it.

Let the current counter value be $k > 0$, and the current state be q , then the following strategy preserves the above invariant.

- If the play is at a state $q = q_{\vee} \in Q_{\vee}$, then there are letters that can be read are of the form a_q from such a state q . Let $(q_{\vee}, q') \in \Delta$ such that there is a word of length $k - 1$ accepted from q' . In this case, \forall reads such an $a_{q'}$ as his next transition, leaving \exists with no non-determinism to resolve.
- If the play is at a state $q = q_{\wedge} \in Q_{\wedge}$, then \forall reads the letter a , which lead to an other state q' chosen by \exists such that there is a transition (q_{\wedge}, q') among the transition of \mathcal{A} . Since there was a transition (q_{\wedge}, q') in the original automaton, it must be the case that there is a word of length $k - 1$ accepted from the copy of the state q' . No matter how \exists resolves the non-determinism on a , she ends up at a state that satisfies the above invariant.

Finally, once the game is at a final state with an empty counter value, we know from the above invariant that we are at a final state. From there, \forall reads the

letter \$. Since the counter value is 0 and the state is a final state, the only transition that is enabled are $(q_F, \$, 0, q_\heartsuit)$ and $(q_F, \$, 0, q_\clubsuit)$, by construction. No matter which of these transitions \exists picks, \forall can respond by playing the other transition and win the game.

\exists wins the letter game on \mathcal{H} if \mathcal{A} is empty If the game stays at q_I forever, \exists wins automatically. If not, after reading a sufficient number of 1s, \forall chooses letter \$. This moves the game to q_0 . Suppose in this run, \forall enters with n as the counter value. Since this automaton accepts no letters by assumption, there is no accepting run of length n from q .

Player \exists uses a strategy that follows the following invariant:

if the letter game is at a state $q \in Q$ for \exists , then there are no words of length equal to the counter value accepted by \mathcal{A} at q

Again, it is true at q_0 . If this invariant is true, when counter value is 0, then the state is not a final state and \forall has to read a \$ to ensure that word is still in the language if at all he needs to win. But \exists can take then the transition $(q, \$, 0, q_{\text{win}}^1)$ to q_{win}^1 , and then read \heartsuit or \clubsuit .

Now we can prove the invariant.

- If the play is at a state $q_\vee \in Q_\vee$, then no matter what letters \forall proposes, there is no non-determinism to resolve for \exists . If
 - \forall reads an a or any $a_{q'}$ such that q' is not adjacent to q , then \exists moves to q_{win}^1 ;
 - \forall reads a \$, then \exists moves to q_{win}^2 ;
 - \forall reads $a_{q'}$ with (q_\vee, q') being a transition in \mathcal{A} , then the play moves to q' , on subtracting 1 but this is a state from which there is no run of length $k - 1$, preserving the invariant.
- If the play is at a state $q = q_\wedge \in Q_\wedge$, then the letters that can be read that are of the form a , which lead to another state q' such that there is a transition (q_\wedge, q') among the transition of \mathcal{A} . If \forall reads anything that is of the form $a_{q'}$, then \exists goes to q_{win} . But if not, since there is at least one transition (q_\wedge, q') in the original automaton such that there are no words of length $k - 1$ accepting from such a state q' , \exists picks that transition in the letter game continuing her play.

This shows that \mathcal{H} is history-deterministic if and only if \mathcal{A} is empty.

B Appendix for Section 4

B.1 Proof of Lemma 18

Proof. We assume the history-deterministic OCN \mathcal{N} is such that it satisfies semilinear-strategy property. Suppose, for each transition δ , the set \mathcal{S}_δ is an eventually periodic set with its period as P_δ , with the maximum number in the preperiodic part as I_δ . Let $I = \max\{I_\delta\}_{\delta \in \Delta}$, and $P = \prod_{\delta \in \Delta} P_\delta$. Thus, each set

\mathcal{S}_δ for each transition δ can be expressed an eventually periodic set with period P , and all numbers in the preperiodic part less than I .

We first construct a non-deterministic one-counter automata \mathcal{B} that accepts the same language as \mathcal{N} . Intuitively, the automaton \mathcal{B} is constructed such that the state space of the automaton stores in its memory, the period and the initial block of the semi-linear sets. The idea is that this automaton's runs would be in bijection with the runs in the net \mathcal{N} that take only good transitions. However, the counter values are 'scaled down' to only remember how many periods have passed, while counter value 0 indicates that the counter value in the original run would have been at most $I+P$. The exact value of the counter value in a run of \mathcal{N} can be inferred as a function of the state space. Formally, $\mathcal{B} = (Q', \Sigma, \Delta', q'_0, F')$, where the set of states Q' contains two types of states. One which encodes the initial block along with the current state and the other which encodes the information corresponding to the repeating block.

More formally, it is given by

$$Q' = \{\langle q, m \rangle \mid q \in Q \text{ and } 0 \leq m \leq I\} \cup \{[q, n] \mid q \in Q \text{ and } 1 \leq n \leq P\}.$$

The set of transitions Δ' is the union of the following sets :

1. $\{(\langle q, i \rangle, \text{zero}, a, 0, [q', j]) \mid (q, i) \xrightarrow{a,d} (q', j+I) \text{ is a good transition in } \mathcal{N}\}$
2. $\{(\langle q, i \rangle, \text{zero}, a, 0, \langle q', j \rangle) \mid (q, i) \xrightarrow{a,d} (q', j) \text{ is a good transition in } \mathcal{N}\}$
3. $\{([q, i], \text{zero}, a, 0, \langle q', j \rangle) \mid (q, i+I) \xrightarrow{a,d} (q', j) \text{ is a good transition in } \mathcal{N}\}$
4. $\{([q, i], X, a, 0, [q', j]) \mid (q, i+I) \xrightarrow{a,d} (q', j+I) \text{ is a good transition in } \mathcal{N}\}$
5. $\{([q, i], X, a, 1, [q', j]) \mid (q, i+I) \xrightarrow{a,d} (q', j+I+P) \text{ is a good transition in } \mathcal{N}\}$
6. $\{([q, i], \neg\text{zero}, a, -1, [q', j]) \mid (q, i+I+P) \xrightarrow{a,d} (q', j+L) \text{ is a good transition in } \mathcal{N}\}$

Here X is a symbol in $\{\text{zero}, \neg\text{zero}\}$. The initial state is $q'_0 = \langle q_0, 0 \rangle$, and the set of final states F' is given by $F' = \{\langle q, m \rangle \mid q \in F \text{ and } 0 \leq m \leq I\} \cup \{[q, n] \mid q \in F \text{ and } 1 \leq n \leq P\}$. We note that any run in \mathcal{B} starting at $q'_0 = \langle q_0, 0 \rangle$ only reaches a state $\langle q, i \rangle$ with counter value 0, where $q \in Q, i < I$. This is because all transitions that go to such a state $\langle q, i \rangle$ test for 0. We define the set $\mathcal{C}'(\mathcal{B}) = \mathcal{C}(\mathcal{B}) \setminus \{(\langle q, i \rangle, p) \mid p > 0, \langle q, i \rangle \in Q'\}$, as a subset of the configuration of \mathcal{B} , which we call *valid configurations* of \mathcal{B} . Any configuration of \mathcal{B} that is not valid cannot be reached.

We show that the runs in the automaton \mathcal{B} are in bijection with the runs in \mathcal{N} that take only good transitions. First, we define a bijection between the valid configuration of \mathcal{B} and the configurations of \mathcal{N} , given by $\Psi : \mathcal{C}'(\mathcal{B}) \rightarrow \mathcal{C}(\mathcal{N})$.

$$\Psi(\alpha) = \begin{cases} (q, i) & \text{if } \alpha = (\langle q, i \rangle, 0), q \in Q \text{ and } 0 \leq i \leq I \\ (q, i+I+c \cdot P) & \text{if } \alpha = ([q, i], c), q \in Q \text{ and } 1 \leq i \leq p \text{ and } c \geq 0 \end{cases}$$

The function Ψ is a bijection, as can be seen by the function $\Theta : \mathcal{C}(\mathcal{N}) \rightarrow \mathcal{C}'(\mathcal{B})$, which is the inverse of Ψ .

$$\Theta((q, i)) = \begin{cases} (\langle q, i \rangle, 0) & \text{where } q \in Q \text{ and } 0 \leq i \leq I \\ ([q, j], c) & \text{where } i = I + j + c \cdot P, q \in Q \text{ and } 1 \leq j \leq p \text{ and } c \geq 0 \end{cases}$$

We note that the transitions in \mathcal{B} are in bijection with good transitions in \mathcal{N} , as $\alpha \xrightarrow{a} \beta$ is a transition in Δ' if and only if $\Psi(\alpha) \xrightarrow{a} \Psi(\beta)$ is a good transition in \mathcal{N} , by construction of Δ' . Thus, we can extend this bijection to get an one-to-one correspondence between runs in \mathcal{B} and runs that take only good transitions in \mathcal{N} . As both Ψ and Θ preserves acceptance of configurations, we get that $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{N})$.

For any accepting word w , any run of \mathcal{B} on the word w corresponds to a run of \mathcal{N} on w that takes only good transitions. By Lemma 15, such a run on \mathcal{N} must exist as the automata is history-deterministic, and it must end in an accepting state of \mathcal{N} , which implies the corresponding run in \mathcal{B} must be accepting as well. Thus, any run of \mathcal{B} on an accepting word in \mathcal{N} must be an accepting run in \mathcal{B} , showing $\mathcal{L}(\mathcal{N}) \subseteq \mathcal{L}(\mathcal{B})$, and hence $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{B})$.

Now, the deterministic one-counter automaton \mathcal{D} , obtained by simply deleting any minimal set of transitions from \mathcal{B} to make it deterministic would accept the same language as \mathcal{N} . This follows from the above paragraph, as any run of \mathcal{B} on an accepting word (in \mathcal{N} or in \mathcal{B}) is accepting.

B.2 Proof of Lemma 19

Proof. Let $\mathcal{N} = (Q, \Sigma, \Delta, q_0, F)$, and let $\gamma = (p, a, e, p')$ be a given transition in Δ . Note that we use e here to denote the change so as to not confuse ourselves with d which we will use to denote change in other transitions.

We would like to show that the set

$$\mathcal{S}_\gamma = \{k | (p, k) \xrightarrow{a, e} (p', k + e) \text{ is a good transition}\}$$

is semilinear. Note that $(p, k) \xrightarrow{a} (p', k + e)$ is a good transition if and only if \exists wins G_1 from $((p, k), (p, k))$ with the following restrictions in the first round of the play :

1. The player \forall picks a as the letter in the first round
2. The player \exists picks γ as the transition on (p, k) , resulting in the transition $(p, k) \xrightarrow{a, e} (p', k + e)$. If \exists is unable to pick γ (due to $k + e$ being negative), then \exists loses immediately.

First we shall construct a simulation game $\mathcal{G}(\mathcal{M}_\gamma, (s, k) \longleftrightarrow \mathcal{M}'_\gamma, (s', k))$ between nets \mathcal{M}_γ and \mathcal{M}'_γ , where s and s' are states in \mathcal{M}_γ and \mathcal{M}'_γ respectively, such that \exists wins the simulation game $\mathcal{G}(\mathcal{M}_\gamma, (s, k) \longleftrightarrow \mathcal{M}'_\gamma, (s', k))$ if and only if $(p, k) \xrightarrow{a, e} (p', k')$ is a good transition in \mathcal{N} . The construction of \mathcal{M}_γ and \mathcal{M}'_γ is similar to that of the nets \mathcal{M} and \mathcal{M}' in Lemma 11, while slightly altering the initial transitions so as to match the above G_1 game with the mentioned restrictions in the first round of the play :

Construction of \mathcal{M}'_γ : The net \mathcal{M}'_γ is essentially the net \mathcal{N} , along with two additional states : s' and $p_\#$, and an additional letter $\#$. The state s' has exactly one outgoing transition (s', a, e, p') . Recall that p' is the target state of the transition

γ . This is to captures \exists only being able to take γ in round 0 of G_1 . From each state p which was accepting in \mathcal{N} , we add the transition $(p, a, 0, p_\#)$ to $p_\#$, and we add self loops on $q_\#$ at $\Sigma_\# = \Sigma \cup \#$. Formally, let $\mathcal{M}'_\gamma = (Q'_\gamma, \Sigma_\#, \Delta', q_0, F')$, where

- The set of states is $Q' = Q \cup \{s', p_\#\}$
- The alphabet $\Sigma_\# = \Sigma \cup \#$
- The accepting states F' is the singleton set $\{q_\#\}$
- The set of transitions Δ' is the union of the following sets
 1. Δ , the set of transitions in \mathcal{N}
 2. $\{(s', a, e, p')\}$
 3. $\{(q, \#, 0, q_\#) \mid q \in F\}$
 4. $\{(q_\#, a, 0, q_\#) \mid a \in \Sigma_\#\}$

Construction of \mathcal{M}_γ : We construct \mathcal{M}_γ to contain an initial state s , which captures \forall only being able to choose the letter $a \in \Sigma$ in the first round of the play in G_1 , along with approximately $(|\Sigma| + 1)$ copies of the states in \mathcal{N} . These copies store the previous read letter in the state space of OCN. The transitions of \mathcal{N} mimic a ‘one-step lag’ in \mathcal{M}_γ . On reading an alphabet, the automaton \mathcal{M}_γ takes a transition in the projection to \mathcal{N} in the first component based on the letter stored in the second component. Note that this transition in the \mathcal{N} component is not based on the current letter. This current letter however is now stored in the state in the destination. Formally, $\mathcal{M}_\gamma = (Q_\gamma, \Sigma_\#, \Delta_\gamma, s, F_\gamma)$, where

- $Q_\gamma = (Q \times \Sigma) \cup \{s\} \cup (F \times \{\#\})$
- s is the initial state
- F_γ , the set of final states is $F \times \{\#\}$
- The set of transitions Δ_γ is the union of the following sets :
 - $\{s \xrightarrow{a,0} (p, a)\}$. This corresponds to \forall only being able to pick a in the first round of G_1 .
 - $\{((q, a), b, d, (q', b)) \mid (q, a, d, q') \in \Delta\}$
 - $\{((q, a), \#, d, (q', \#)) \mid (q, a, d, q') \in \Delta \text{ and } q \in F\}$

We claim that $(p, k) \xrightarrow{a,e} (p, k + e)$ is a good transition if and only if \exists wins the simulation game $\mathcal{G}(\mathcal{M}_\gamma, (s, k) \hookrightarrow \mathcal{M}'_\gamma, (s', k))$. Note that $(p, k) \xrightarrow{a,e} (p, k + e)$ is a good transition if and only if \exists wins the game G_1 from $((p, k), (p, k))$ with the restrictions in the first round mentioned above. Using an argument almost identical to that of in Lemma 11, we can show that \exists wins the game G_1 with these restrictions in the first round if and only if \exists wins the simulation game $\mathcal{G}(\mathcal{M}_\gamma, (s, k) \hookrightarrow \mathcal{M}'_\gamma, (s', k))$. As the set of such k 's is semilinear by Theorem 12, we get that \mathcal{S}_γ is semilinear as well.

B.3 Proof of Lemma 22

Proof (Proof of Lemma 22). Given two history-deterministic DOCN $\mathcal{H} = (Q_A, \Sigma, \Delta_A, q_A^0, F_A)$ and $\mathcal{H}' = (Q_B, \Sigma, \Delta_B, q_B^0, F_B)$. Note that we can assume that \mathcal{H}' accepts at least one element that is not in \mathcal{H} . This can be done because we can always consider the following OCN \mathcal{M} such that for some symbol $\boxtimes \notin \Sigma$, defined as follows :

$$\mathcal{M} = (Q'_B, \Sigma \cup \{\boxtimes\}, \Delta'_B, q_B^0, F'_B),$$

where for a new element q_* not in Q ,

- the set of states $Q'_B = Q_B \cup \{q_*\}$,
- the set of transitions $\Delta'_B = \Delta_B \cup \{(q_0, \boxtimes, 0, q_*)$, and
- the final states $F'_B = F_B \cup \{q_*\}$.

The automaton \mathcal{M} is history-deterministic, and the language accepted by \mathcal{M} , is $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{H}') \cup \{\boxtimes\}$. Note that $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{H}')$ if and only if $\mathcal{L}(\mathcal{H}) \subsetneq \mathcal{L}(\mathcal{M})$.

Henceforth, we will only consider such automaton \mathcal{H} , \mathcal{H}' where \mathcal{H}' accepts the word \boxtimes which is not accepted by \mathcal{H} , such that they are both history-deterministic.

We an OCN \mathcal{N} , which is history-deterministic if and only if $\mathcal{L}(\mathcal{H}) \subset \mathcal{L}(\mathcal{H}')$. Let $\mathcal{N} = (Q_N, \Sigma \cup \{\heartsuit\}, \Delta_N, q_N^0, F_N)$, where

- the set of states $Q_N = Q_A \cup Q_B \cup \{q_N^0\}$,
- the set of transitions $\Delta_N = \Delta_A \cup \Delta'_B \cup \{q_H^0, \heartsuit, 0, q_A^0\} \cup \{(q_H^0, \heartsuit, 0, q_B^0)\}$, and
- the final states $F_N = F_A \cup F_B$.

Suppose \mathcal{N} constructed as above is history-deterministic, there is resolver on q_N^0 , that chooses the transition $q_N^0 \xrightarrow{\heartsuit} q_A^0$ or $q_N^0 \xrightarrow{\heartsuit} q_B^0$. Since by our assumption, the language of \mathcal{H}' contains a word which is not in the language of \mathcal{H} , if the resolver did not choose transition to q_B^0 , then \exists loses the letter game as \forall can give as input this word not in the language of \mathcal{H} , but in \mathcal{H}' . Therefore, any resolver must choose the transition to the copy of \mathcal{H}' . This implies that for any word accepted by \mathcal{H} , the resolver has a strategy henceforth to produce a sequence of transitions in \mathcal{H}' that lead to an accepting state, implying that $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{H}')$.

If $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{H}')$, the resolver only needs to deal with non-determinism in the first step. Choosing the transition $q_N^0 \xrightarrow{\heartsuit} q_B^0$ at q_N^0 ensures \exists wins the letter game on \mathcal{N} , since \mathcal{H}' is history-deterministic.

Since the obtained automaton \mathcal{N} has size linear in \mathcal{H} and \mathcal{H}' we can check history-determinism of \mathcal{N} to decide the inclusion $\mathcal{L}(\mathcal{H}) \subseteq \mathcal{L}(\mathcal{H}')$.

B.4 Proof of Lemma 23

Proof (Proof of Lemma 23). Let us first show that given two history-deterministic one-counter nets \mathcal{M} and \mathcal{N} ,

- (\Rightarrow) \mathcal{M} simulates \mathcal{N} then $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{N})$ and
- (\Leftarrow) $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{N})$, then \mathcal{M} simulates \mathcal{N} .

\Rightarrow : Suppose \mathcal{M} simulates \mathcal{N} . Then, over any accepting word $w \in \mathcal{L}(\mathcal{M})$, there is an accepting run ρ of it in \mathcal{M} , and as \mathcal{M} simulates \mathcal{N} , the run corresponding to ρ in \mathcal{N} must be accepting as well. Thus, $w \in \mathcal{L}(\mathcal{N})$.

\Leftarrow : Suppose $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{N})$. Then, the player \exists wins the simulation game $\mathcal{G}(\mathcal{M} \hookrightarrow \mathcal{N})$: The player \exists can simply ignore \forall 's run in \mathcal{M} , and play according to her letter game strategy in \mathcal{N} . If \forall 's token is at an accepting state at the end of any round in the game after having read w , then $w \in \mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{N})$, which implies \exists 's token must be on an accepting state as well, as \exists was playing according to her letter game strategy in \mathcal{N} .

Note that Σ^* can be given by a one state finite state automata.

The problem of universality reduces to checking if simulation between a one-state finite automaton and the input net \mathcal{M} which is in **P** from the results of Kucera (Lemma 2, [24]).

C Appendix for Section 5

C.1 Proof of Lemma 26

Proof. Let us describe what reachability games on succinct one-counter nets (SOCN) are [20,22]. The arena of a SOCN-reachability game $\mathcal{G}(\mathcal{N})$ consists of a one-counter net $\mathcal{N} = (Q, \{a\}, \Delta, q_0, F)$ over an unary alphabet. However, the states are partitioned as $Q = Q_{\vee} \uplus Q_{\wedge}$ among the players \vee and \wedge respectively, such that any transition is only between a \wedge and \vee state or a \vee and \wedge state. If the play is at a \vee (\wedge) state, then the player \vee (\wedge) chooses a transition at that state to go to the next configuration. Moreover, these transitions are allowed to increment and decrement more than 1, and can be any arbitrary value $d \in \mathbb{Z}$. The starting state of the game is the configuration $(q_0, 0)$. We consider the problem of reachability to a configuration $(q_F, 0)$ for some $q_F \in F$.

The version of SOCN-reachability game we have defined is slightly different from the version considered in the theorem statement of Hunter [20], but one can see that these can be shown to be inter-reducible [22].

The decision problem, which we call SOCN-GAME is

Given: A SOCN-reachability game $\mathcal{G}(\mathcal{N})$ such that the counter change in transitions are encoded in binary.

Question: Does there exist a winning strategy for \vee in $\mathcal{G}(\mathcal{N})$.

As mentioned, the above problem is **EXPSpace**-complete [20,22]. We show the **EXPSpace**-hardness for deciding history-determinism using the above problem.

The net \mathcal{M} is designed such that a transcription of any play on the succinct one-counter game is an accepting word.

The following introduction to the construction is best read referring to the rigorous construction that follows it.

Describing the states of the automaton: To be able to recreate the transcript of a play of the SOCN-reachability game in the letter game of \mathcal{M} , the state space contains

1. A ‘main’ copy of the states of \mathcal{N} , in which the game would stay if \forall had a strategy to win. These are used to maintain a run on \mathcal{N} .
2. A resolution copy of states of Q_\wedge for \exists to stay in, until the non-determinism chosen by \exists is faithfully re-played by \forall .
3. A ‘copy’ of \mathcal{N} to ensure that any transcript of a run that is encoded indeed is a real run on the one-counter automaton \mathcal{N} . For each q and each transitions of \mathcal{N} , we add states to the set Q_{win} . These states are called so, because from here, \exists can win the letter game. There is no non-determinism to resolve.
4. A few extra states, to preserve the winner in the succinct reachability game. These ensure that if a play of the letter game in \mathcal{M} corresponds to a winning transcript of \wedge player, then \exists can go to a state with no non-determinism. However, there are also states q_\heartsuit and q_\clubsuit which are the states that \forall would be able to make \exists reach if there is a winning play for the \vee player.

Describing the alphabet of the automaton: The alphabet contains an input a_δ for each transition from a \vee state. This is to make sure \forall , who picks the letter in the letter game is in charge of picking the next transition for the player \vee in the game. There is also a single letter a which creates non-determinism such that in the letter game, \exists can resolve the non-determinism by picking the next transition. Later on, to ensure a fair play, \forall is forced to confirm this non-determinism by reading the letter a_δ corresponding to the transition that \exists had chosen. If \forall picks a different transition, then the play moves to the component which we call Q_{win} from which all transcriptions of sequences that are ‘valid’ in the original automaton are accepting, and there is no non-determinism, making it winning for \exists in the letter game.

There are also some special symbols used in the following way:

- $\$$ is used by \forall to indicate he is at a final state with 0 in the counter. If he reads it anywhere else, \exists wins the letter game trivially.
- \heartsuit and \clubsuit are to be read immediately after $\$$, but the states q_\heartsuit can only read \heartsuit and q_\clubsuit can only read \clubsuit .

Language accepted by the automaton: The language accepted by the automaton would be any prefix of the words of the form $a_{\delta_0} a a_{\delta_1} a_{\delta_2} a a_{\delta_3} a_{\delta_4} \dots a a_{\delta_k} a_{\delta_{k+1}}$ followed by a $\$$ then one of \heartsuit or \clubsuit such that $\delta_0 \delta_1 \delta_2 \dots \delta_k \delta_{k+1}$ is a valid sequence of transitions in \mathcal{N} . Note that the above words essentially is a sequence of letters of the form a_δ transitions, but with a s read before reading a ‘transition’ from a \wedge state.

Transitions of the automaton: Now we intuitively describe the transitions of the automaton that accept such a language above. They are constructed so that reading each a_δ increments or decrements the counter by the same amount prescribed by δ .

From the ‘main’ copy of the state belonging to \vee , the transitions are such that if there was a transition from this state in \mathcal{N} , a copy of the transition also is added. Moreover, there are specific letters that one can read to go to the next state prescribed by the transition. For \wedge however, these transitions are labelled by a , and perform no increment or decrement. They instead take the run to

a temporary copy of the state, from which \forall can read the a_δ corresponding to the delta that \exists had chosen. If the letter does not correspond to the same transition that \exists had picked, then the run move to the copy of the game Q_{win} , which just ensures that \exists wins.

More formally, given a game on \mathcal{N} , we define $\mathcal{M} = (Q', \Sigma', \Delta', q_0, F')$. Before defining \mathcal{M} however, we first describe the Q_{win} , a sub-net, which will form a part of the main automaton. This is done so as to make sure the main definition has less clutter.

The Q_{win} gadget: We describe the set of states Q_{win} and the transitions associated with it in more detail. Recall that this part of the net is to mainly ensure that only valid transcripts of a run on \mathcal{N} are the ones that are accepted. For this, we essentially take one copy of Q and one more copy of Q_Δ . We add one more state to recognise that the transcript has ended if a $\$$ has been read.

We therefore have, $Q_{\text{win}} = Q \uplus Q_\Delta \uplus \{q_\$ \}$. We call states q_{win} with the subscript if q is from the copy of Q and q_{twinn} for a copy of $q \in Q_\Delta$. There are the following transitions:

- $(q_{\text{win}}, a_\delta, d, p_{\text{win}})$ for all $q \in Q_\forall$ and $\delta = (q, d, p)$, a transition in \mathcal{N} ;
- $(p_{\text{win}}, a, 0, p_{\text{twinn}}^{\text{win}})$ for all $p \in Q_\Delta$
- $(p_\delta^{\text{win}}, a_\delta, d, q)$ for all $p \in Q_\Delta$ and $\delta = (p, d, q)$, a transition in \mathcal{N} ;
- $(q, \$, 0, q_\$)$ for all $q \in Q$.

We describe the automaton $\mathcal{M} = (Q', \Sigma', \Delta', q_0, F')$ where,

- the set of states $Q' = Q \cup \{q_\delta \mid \delta \in \Delta\} \cup \{q_\$ \} \cup \{q_\heartsuit, q_\clubsuit, q_{\text{last}}\}$, and the states Q_{win}
- the alphabet set is $\Sigma = \{\$, \heartsuit, \clubsuit, \}$ $\cup \{a_\delta \mid \delta \in \Delta\} \cup \{a\}$
- the start state is q_0 , which is copy of the start state at Q , and
- all states of Q' are final states, including Q_{win}

The set of transitions are the union of the sets of transitions below, along with those of Q_{win} and some defined from the states to Q_{win} and one transitions back from it to q_{last} to end the word.

1. $\{(q_\forall, a_q, d, q) \mid (q_\forall, d, q) \in \Delta \text{ and } q_\forall \in Q_\forall\}$
2. $\{(p, a, 0, p_\delta) \mid \delta = (p, d, q) \in \Delta \text{ and } p \in Q_\Delta\}$
3. $\{(p_\delta, a_\delta, d, q) \mid \delta = (p, d, q) \in \Delta \text{ and } p \in Q_\Delta\}$
4. $\{(p_\delta, a_{\delta'}, d', q'_{\text{win}}) \mid \delta = (p, d, q) \text{ and } \delta' = (p, d', q'), \delta \neq \delta' \text{ and } p \in Q_\Delta\}$
5. $\{(q_F, \$, -1, q_\$), (q_F, \$, 0, q_\clubsuit), (q_F, \$, 0, q_\heartsuit) \mid q_F \in F\}$
6. $\{(q, \$, 0, q_\$) \mid q \notin F\}$
7. $\{(q_\heartsuit, \heartsuit, 0, q_{\text{last}}), (q_\clubsuit, \clubsuit, 0, q_{\text{last}})\}$
8. From Q_{win} we have $(q_\$, \heartsuit, 0, q_{\text{last}}), (q_\$, \clubsuit, 0, q_{\text{last}})\}$

Observe that transitions described in items 4, 5, 6 involve transitions to Q_{win} . In item 4., note that both the transitions δ and δ' should be from the same state for this transition to exist.

We now proceed to showing that the above construction is such that

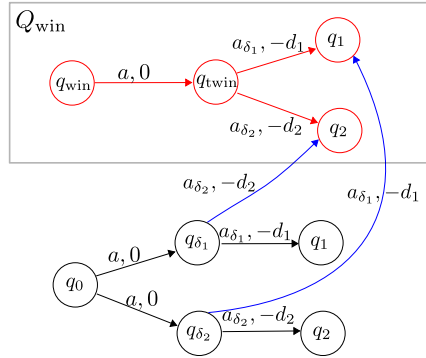


Fig. 5. A snapshot of the Q_{win} gadget with the states in Q_{win} in red, with the ‘main’ vertices which are not in Q_{win} in black. If the non-determinism in the run of the succinct OCN game’s imitation here is incorrectly resolved anywhere then \exists can take transitions to the red states from which there is no non-determinism.

- \Rightarrow If the reachability game on succinct net \mathcal{N} is won by \vee , then \mathcal{M} constructed is not history-deterministic and \forall can win the letter game
- \Leftarrow If the reachability game on \mathcal{N} is won by \wedge , then \exists has a strategy to win the letter game, and \mathcal{M} is history-deterministic

Winning for \vee reachability game implies winning for \forall in letter game Here we prescribe \forall ’s strategy which is essentially to follow the reachability strategy of \vee . When the game is at a state

- $q \in Q_{\vee}$ with counter value k , then he picks a_{δ} such that the transition δ ensures $(q, k) \xrightarrow{d} (q', k')$ is a winning transition prescribed by a fixed strategy in the game. This leads to no non-determinism.
- $q \in Q_{\wedge}$ with counter value k , then he picks a , but no matter which transition in the game \exists picks, she reaches a configuration that is still winning for \vee in the succinct game, this is because any $(q, k) \xrightarrow{d} (q', k')$ was a winning transition in the game.

This strategy maintains an invariant that if the play of a letter game was at a configuration such that the corresponding configuration in \mathcal{N} was winning for \vee , then \forall can ensure that in the letter game, any transition that \exists picks also leads to a configuration where this is true.

Since the game is winning from $(q_0, 0)$, we know that eventually \forall would reach a state that is a final state with a counter value 0. Once he reaches such a configuration, he reads $\$$. This ensures that only the transitions $(q_F, \$, 0, q_{\heartsuit})$ or $(q_F, \$, 0, q_{\clubsuit})$ are enabled. From here, whichever transition \exists picks, he reads the other letter corresponding to it to win.

Winning for \wedge in reachability game implies \exists wins letter game \exists ’s strategy is also to mimic the strategy in the underlying reachability game. She fixes such a

strategy. When the game is at a configuration (q, k) in the net \mathcal{N} she does the following:

- for $q \in Q_\vee$ with counter value k , if \forall picks a_δ such that δ ensures $(q, k) \xrightarrow{d} (q', k')$ was a transition, then \exists needs to make no decisions. If not, the game proceeds to the copy Q_{win} and we can show that any sequence of runs that is played that has a valid run is winning for the player \exists anyway. If the play instead stays in the ‘main’ copy, then the new configuration reached maintains the invariant.
- for $q \in Q_\wedge$ with counter value k , if \forall picks a , then \exists picks a the transitions δ corresponding to the configuration prescribed by her winning strategy. Note that later if \forall does not pick a_δ , then \exists wins by going to Q_{win} . Observe that if the transition prescribed is such that the run goes below 0, that run turns out to be not accepting because of the gadget described. This means \forall loses the letter game again immediately. If the transition still stays above 0, then the configuration proceeds to a (q', k') , prescribed by \wedge ’s strategy in the reachability game to avoid visiting $(q_F, 0)$. The position (q', k') is such that there is no winning strategy for \forall from it.

Observe that from the invariant, \exists never reaches $(q_F, 0)$ and therefore she never has to resolve the non-determinism that occurs at q_F to states q_\heartsuit and q_\clubsuit . Any infinite play is also won by \exists , and therefore she wins the letter game on \mathcal{M} .

C.2 Proof of Theorem 28

Proof. Consider the following problem :

DOCA Inclusion: Given two DOCA \mathcal{A} and \mathcal{B} , is $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$?

We reduce DOCA inclusion to the problem of deciding whether a given One-counter automaton is history-deterministic. Valiant, in Section 5.1 of his thesis [30] shows that the DOCA inclusion problem is undecidable [31]. This immediately shows that the problem of deciding if a given one-counter automaton is history-deterministic is also undecidable.

Note that this construction is similar to the one in the construction of Lemma 22. We nevertheless re-state it here, for completeness.

Given DOCA $\mathcal{A} = (Q_A, \Sigma, \Delta_A, q_A^0, F_A)$ and $\mathcal{B} = (Q_B, \Sigma, \Delta_B, q_B^0, F_B)$, consider the automaton \mathcal{B}' , for some symbol $\boxtimes \notin \Sigma$, defined as follows :

$$\mathcal{B}' = (Q'_B, \Sigma \cup \{\boxtimes\}, \Delta'_B, q_B^0, F'_B),$$

where for a new element q_* not in Q ,

- the set of states $Q'_B = Q_B \cup \{q_*\}$,
- the set of transitions $\Delta'_B = \Delta_B \cup \{(q_0, \text{zero}, \boxtimes, 0, q_*), (q_0, \neg\text{zero}, \boxtimes, 0, q_*)\}$,
and
- the final states $F'_B = F_B \cup \{q_*\}$.

The automaton \mathcal{B}' is deterministic, and the language accepted by \mathcal{B}' , is $\mathcal{L}(\mathcal{B}') = \mathcal{L}(\mathcal{B}) \cup \{\boxtimes\}$. Note that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ if and only if $\mathcal{L}(\mathcal{A}) \subsetneq \mathcal{L}(\mathcal{B}')$.

We now describe the automaton \mathcal{H} , which is history-deterministic if and only if $L(\mathcal{A}) \subset L(\mathcal{B}')$. Let $\mathcal{H} = (Q_H, \Sigma \cup \{\boxtimes, \heartsuit\}, \Delta_H, q_H^0, F_H)$, where

- the set of states $Q_H = Q_A \cup Q'_B \cup \{q_H^0\}$,
- the set of transitions $\Delta_H = \Delta_A \cup \Delta'_B \cup \{q_H^0 \xrightarrow{\heartsuit} q_A^0\} \cup \{q_H^0 \xrightarrow{\heartsuit} q_B^0\}$, and
- the final states $F_H = F_A \cup F'_B$.

Suppose \mathcal{H} constructed as above is history-deterministic, there is resolver on q_H^0 , that chooses the transition $q_H^0 \xrightarrow{\heartsuit} q_A^0$ or $q_H^0 \xrightarrow{\heartsuit} q_B^0$. Since the language of \mathcal{B}' contains the word \boxtimes , which is not in the language of \mathcal{A} , if the resolver did not choose transition $q_H^0 \xrightarrow{\heartsuit} q_B^0$, then \exists loses the letter game if \forall gave as input \boxtimes . Therefore, any resolver must choose the transition to the copy of \mathcal{B}' . This implies that for any word accepted by \mathcal{A} , the resolver has a strategy henceforth to produce a sequence of transitions in \mathcal{B}' , implying that this word must also be accepted by \mathcal{B} . Hence we have $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$.

If $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, the resolver only needs to deal with non-determinism in the first step. Choosing the transition $q_H^0 \xrightarrow{\heartsuit} q_B^0$ at q_H^0 ensures \exists wins the letter game and hence \mathcal{H} is history-deterministic.