Abstract

Register automata are finite automata equipped with a finite set of registers in which they can store data, i.e., elements from an unbounded or infinite alphabet. They provide a simple formalism to specify the behaviour of reactive systems operating over data $\omega$-words. We study the synthesis problem for specifications given as register automata over a linearly ordered data domain (e.g., $(\mathbb{N}, \leq)$ or $(\mathbb{Q}, \leq)$), which allow for comparison of data with regards to the linear order. To that end, we extend the classical Church synthesis game to infinite alphabets: two players, Adam and Eve, alternately play some data, and Eve wins whenever their interaction complies with the specification, which is a language of $\omega$-words over ordered data. Such games are however undecidable, even when the specification is recognised by a deterministic register automaton. This is in contrast with the equality case, where the problem is only undecidable for nondeterministic and universal specifications.

Thus, we study one-sided Church games, where Eve instead operates over a finite alphabet, while Adam still manipulates data. We show they are determined, and deciding the existence of a winning strategy is in $\text{ExpTime}$, both for $\mathbb{Q}$ and $\mathbb{N}$. This follows from a study of constraint sequences, which abstract the behaviour of register automata, and allow us to reduce Church games to $\omega$-regular games. Lastly, we apply these results to the transducer synthesis problem for input-driven register automata, where each output data is restricted to be the content of some register, and show that if there exists an implementation, then there exists one which is a register transducer.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Logic and verification; Theory of computation $\rightarrow$ Automata over infinite objects; Theory of computation $\rightarrow$ Transducers

Keywords and phrases Synthesis, Church Game, Register Automata, Transducers, Ordered Data Words

Digital Object Identifier 10.4230/LIPIcs...
Synthesis is the problem of automatically constructing a system from a behavioral specification. It was first proposed by Church as a game problem: two players, Adam in the role of the environment and Eve in the role of the system, alternately pick the values from alphabets \( I \) and \( O \). Adam starts with \( i_0 \in I \), Eve responds with \( o_0 \in O \), ad infinitum. Their interaction results in the infinite outcome \( i_0 o_0 i_1 o_1 ... \in (I \cdot O)^\omega \). The winner is decided by a winning condition, represented as a language \( S \subseteq (I \cdot O)^\omega \) called specification: if the outcome of Adam and Eve’s interaction belongs to \( S \), the play is won by Eve, otherwise by Adam. Eve wins the game if she has a strategy \( \lambda_E : I^+ \to O \) to pick values, depending on what has been played so far, allowing her to win against any Adam strategy. Similarly, Adam wins the game if he has a strategy \( \lambda_A : O^* \to I \) to win against any Eve strategy. In the original Church problem, the alphabets \( I \) and \( O \) are finite, and specifications are \( \omega \)-regular languages. The seminal papers [12, 34] connected Church games to zero-sum games on finite graphs. They also showed that Church games enjoy the property of determinacy: every game is either won by Eve or otherwise by Adam, and finite-memoriness: if Eve wins the game then she can win using a finite-memory strategy which can be executed by e.g. Mealy machines.

The synthesis and Church games were extensively studied in many settings, for example, quantitative, distributed, non-competitive, yet Adam and Eve usually interact via finite alphabets. But real-life systems often operate values from a large to infinite data domain. Examples include data-independent programs [40, 24, 31], software with integer parameters [10], communication protocols with message parameters [15], and more [9, 38, 14]. To address this challenge, recent works looked at synthesis where infinite-alphabet specifications are described by register automata and systems (corresponding to Eve strategies in Church games) by register transducers [16, 26, 27, 17].

Register automata extend finite-state automata to infinite alphabets \( \mathcal{D} \) by introducing a finite number of registers [25]. In each step, the automaton reads a data from \( \mathcal{D} \), compares it with the values held in its registers, depending on this comparison it decides to store the data into some of the registers, and then moves to a successor state. This way it builds a sequence of configurations (pairs of state and register values) representing its run on reading a word from \( \mathcal{D}^\omega \): it is accepted if the visited states satisfy a certain condition, e.g. parity. Transducers are similar except that in each step they also output the content of one register.

Previous synthesis works [16, 26, 27, 17] focused on register automata and transducers operating in the domain \( (\mathcal{D}, =) \) equipped with equality tests only. Related works [21, 30] on synthesis of data systems and which do not rely on register automata are also limited to equality tests or do not allow for data comparison. Thus, we cannot synthesise systems that output the largest value seen so far, grant a resource to a process with the lowest id, or raise an alert when a heart sensor reads values forming a dangerous curve. These tasks require \( \leq \).

We study Church games where Adam and Eve have infinite alphabet \( (\mathcal{D}, \leq) \), namely the dense domain \((\mathbb{Q}, \leq)\) or the nondense domain \((\mathbb{N}, \leq)\), and specifications are given as register automata. Already in the case of infinite alphabets \( (\mathcal{D}, =) \), finding a winner is undecidable when specifications are given as nondeterministic or universal register automata [16, 17], so the works either restricted Eve strategies to register transducers with an a-priori fixed number of registers or considered specifications given as deterministic automata. The case of \((\mathbb{N}, \leq)\) is even harder. Here, Church games are undecidable already for specifications given as deterministic register automata, because they can simulate two-counter machines (Theorem 10). For example, to simulate an increment of a counter, whose value is currently kept in a register \( c \), the automaton asks Adam to provide a data \( d \) above the value \( \nu(c) \) of the counter, saves it into a register \( c_{\text{new}} \), and asks Eve to provide the value between \( \nu(c) \)
and \( r'(c_{nca}) \). If Eve can do this, then Adam cheated and Eve wins, otherwise the game continues. Adam wins if eventually the halting state is reached. However, this proof breaks in the asymmetric setting, where Adam provides data but Eve picks labels from a finite alphabet only. We now give an example to better illustrate the one-sided setting.

**Example.** Figure 1 illustrates a game arena where Adam's states are squares and Eve’s states are circles. Eve’s objective is to reach the top, while Adam tries to avoid it. There are two registers, \( r_M \) and \( r_l \), and Eve’s finite alphabet is \( \{a, b\} \). The test \( \top \) (true) means that the comparison of the input data with the register values is not important, the test \( r_l < * < r_M \) means that the data should be between the values of registers \( r_l \) and \( r_M \), and the test ‘else’ means the opposite. The writing \( \downarrow r \) means that the data is stored into the register \( r \). At first, Adam provides some data \( d_M \), serving as an upper bound stored in \( r_M \). Register \( r_l \), initially 0, holds the last data \( d_l \) played by Adam. Consider state 3: if Adam provides a data outside of the interval \([d_l, d_M]\), he loses; if it is strictly between \( d_l \) and \( d_M \), it is stored into register \( r_l \) and the game proceeds to state 4. There, Eve can either respond with label \( b \) and move to state 5, or with \( a \) to state 3. In state 5, Adam wins if he can provide a data strictly between \( d_l \) and \( d_M \), otherwise he loses. Eve wins this game in \( \mathbb{N} \): for example, she could always respond with label \( a \), looping in states 3-4. After a finite number of steps, Adam is forced to provide a data \( d_M \), losing the game. An alternative Eve winning strategy, that does depend on Adam data, is to loop in 3–4 until \( d_M - d_l = 1 \) (hence she has to memorise the first Adam value \( d_M \)), then move to state 5, where Adam will lose. In the dense domain \( \mathbb{Q} \), however, the game is won by Adam, because he can always provide a value within \([d_l, d_M]\) for any \( d_l < d_M \), so the game either loops in 3–4 forever or reaches state 6.

Despite being asymmetric, one-sided Church games are quite expressive. For example, they enable synthesis of runtime data monitors that monitor the input data stream and raise a Boolean flag when a critical trend happens, like oscillations above a certain amplitude. Another example: they allow for synthesis of register transducers which can output data present in one of the registers of the specification automaton (also studied in [17]). Register-transducer synthesis serves as our main motivation for studying Church games.

The key idea used to solve problems about register automata is to forget the precise values of input data and registers, and track instead the constraints (also called types) describing the relations between them. In our example, all registers start in 0 so the initial constraint is \( r^1_l = r^1_M \), where \( r^i \) abstracts the value of register \( r \) at step \( i \). Then, if Adam provides a data above the value of \( r_l \), the constraint becomes \( r^2_l < r^2_M \) in state 2. Otherwise, if Adam had provided a data equal to the value in \( r_l \), the constraint would be \( r^2_l = r^2_M \). In this way the constraints evolve during the play, forming an infinite sequence. Looping in states 3–4 induces the constraint sequence \( (r^i_l < r^{i+1}_l < r^i_M = r^{i+1}_M)^\omega \). It forms an infinite chain \( r^3_l < r^4_l < ... \) bounded by constant \( r^3_M = r^4_M = ... \) from above. In \( \mathbb{N} \), as it is a well-founded order, it is not possible to assign values to the registers at every step to satisfy all constraints, so the sequence is not satisfiable. Before elaborating on how this information can be used to solve Church games, we describe our results on satisfiability of constraint sequences. This topic was inspired by the work [36] which studies, among others, the nonemptiness problem of constraint automata, whose states and transitions are described by constraints. In particular, they show [36, Appendix C] that satisfiability of constraint sequences can be checked by nondeterministic \( \omega \)-B-automata [4]. Nondeterminism however poses a challenge in synthesis, and it is not known whether games with winning objectives as nondeterministic \( \omega \)-B-automata
are decidable. In contrast, we describe a deterministic max-automaton [7] characterising the satisfiable constraint sequences in \( \mathbb{N} \). As a consequence of [8], games over such automata are decidable. Then we study constraint sequences whose certain chains are bounded, because they happen to be useful for solving Church games with register automata. We show how to assign values to the registers in such a constraint sequence on-the-fly, no matter what the future beholds, in order to satisfy this constraint sequence.

To solve one-sided Church games with a specification given as a register automaton \( S \) for \((\mathbb{N}, \leq)\) and \((\mathbb{Q}, \leq)\), we reduce them to certain finite-arena zero-sum games, which we call feasibility games. The states and transitions of the game are those of the specification automaton \( S \). The winning condition requires Eve to satisfy the original objective of \( S \) only on feasible plays, i.e. those that induce satisfiable constraint sequences. In our example, the play \( 1 \ 2 \ (3 \ 4)^\omega \) does not satisfy the parity condition, yet it is won by Eve in the feasibility game since it is not satisfiable in \( \mathbb{N} \), and therefore there is no corresponding play in the Church game. We show that if Eve wins the feasibility game, then she wins the Church game, using a strategy that simulates the register automaton \( S \) and simply picks one of its transitions. It is also sufficient: if Adam wins the feasibility game then he wins the Church game. To prove this, we construct, from an Adam strategy winning in the feasibility game, an Adam data strategy winning in the Church game. This step uses the previously mentioned result on satisfiability of constraint sequences of a special kind. Overall, our results on one-sided Church games in \((\mathbb{N}, \leq)\) and \((\mathbb{Q}, \leq)\) are:

- they are decidable in time exponential in the number of registers of the specification,
- they are determined: every game is either won by Eve or by Adam, and
- if Eve wins, then she has a winning strategy that can be described by a register transducer with a finite number of states and which picks transitions in the specification automaton.

Finally, these results allow us to solve the register-transducer synthesis problem from input-driven output specifications [17] over ordered data.

**Related works.** [18] studies synthesis from variable automata with arithmetics (we only have \( \leq \)) which are incomparable with register automata; they only consider the dense domain. The paper [19] studies strategy synthesis but, again, mainly in the dense domain. A similar one-sided setting was studied in [20] for Church games with a winning condition given by logical formulas, but only for \((\mathbb{D}, =)\). The work on automata with atoms [29] implies our decidability result for \((\mathbb{Q}, \leq)\), even in the two-sided setting, but not the complexity result, and it does not apply to \((\mathbb{N}, \leq)\). Our setting in \( \mathbb{N} \) is loosely related to monotonic games [2]: they both forbid infinite descending behaviours, but the direct conversion is unclear. Games on infinite arenas induced by pushdown automata [39, 11, 1] or one-counter systems [37, 22] are orthogonal to our games.

**Outline.** We start with Section 2 on satisfiability of constraint sequences, which is the main technical tool, then describe our results on Church games in Section 3 and synthesis in Sect.4.

## 2 Satisfiability of Constraint Sequences

In this paper, \( \mathbb{N} = \{0, 1, \ldots\} \). A data domain \( \mathcal{D} \) is an infinite countable set of elements called *data*, linearly ordered by some order denoted \( < \). We consider two data domains, \( \mathbb{N} \) and \( \mathbb{Q} \), with their usual order. We also distinguish a special element 0 of \( \mathcal{D} \): in \( \mathbb{Q} \) its choice is not important, in \( \mathbb{N} \) it is the expected zero (the minimal element).

**Registers and their valuations.** Let \( R \) be a finite set of elements called *registers*, intended to contain data values, i.e. values in \( \mathcal{D} \). A register valuation is a mapping \( \nu : R \to \mathcal{D} \) (also written \( \nu \in \mathcal{D}^R \)). We write \( 0^R \) to denote the constant valuation \( \nu_0(r) = 0 \) for all \( r \in R \).
Constraint sequences, consistency and satisfiability. Fix a set of registers $R$ (which can also be thought of as variables), and let $R’ = \{ r’ \mid r \in R \}$ be the set of their primed versions. Fix a data domain $D$. In what follows, the symbol $\triangleright$ denotes one of $>$, $<$, or $=$. A constraint is a maximal consistent set of atoms of the form $t_1 \triangleright t_2$ where $t_1, t_2 \in R \cup R’$. It describes how register values change in one step: their relative order at the beginning (when $t_1, t_2 \in R$), at the end (when $t_1, t_2 \in R’$), and between each other (with $t_1 \in R$ and $t_2 \in R’$). E.g., $C = \{ r_1 < r_2, r_1 < r_1’, r_2 > r_2’, r_1’ < r_2’ \}$ is a constraint over $R = \{ r_1, r_2 \}$, which is satisfied, for instance, by the two successive valuations $\nu_0: \{ r_1 \mapsto 1, r_2 \mapsto 4 \}$ and $\nu_1: \{ r_1 \mapsto 2, r_2 \mapsto 3 \}$. However, the set $\{ r_1 < r_2, r_1 > r_1’, r_2 < r_2’, r_1’ > r_2’ \}$ is not consistent.

Given a constraint $C$, the writing $C_{|R}$ denotes the subset of its atoms $r \triangleright s$ for $r, s \in R$, and $C_{|R’}$ — the subset of atoms over primed registers. Given a set $S$ of atoms $r’ \triangleright s’$ over $R’, s’ \in R’$, let $\text{unprime}(S)$ be the set of atoms derived by replacing every $r’ \in R’$ by $r$.

A constraint sequence is an infinite sequence of constraints $C_0 C_1 \ldots$ (when we use finite sequences, we explicitly state it). It is consistent if for every $i$: $\text{unprime}(C_i | R) = C_{i+1} | R’$, i.e., the register order at the end of step $i$ equals the register order at the beginning of step $i + 1$. Given a valuation $\nu \in D^R$, define $\nu’ \in D^R$ to be the valuation that maps $\nu'(r') = \nu(r)$ for every $r \in R$. A valuation $w \in D^{R \cup R’}$ satisfies a constraint $C$, written $w \models C$, if every atom holds when we replace every $r \in R \cup R’$ by $w(r)$. A constraint sequence is satisfiable if there exists a sequence of valuations $\nu_0 \nu_1 \ldots$ in $(D^R)^\omega$ such that $\nu_i \cup \nu_{i+1} \models C_i$ for all $i \geq 0$. If, additionally, $\nu_0 = 0^R$, then it is 0-satisfiable. Notice that satisfiability implies consistency.

Examples. Let $R = \{ r_1, r_2, r_3, r_4 \}$. Let a consistent constraint sequence $C_0 C_1 \ldots$ start with $\{ r_1 < r_2 < r_3 < r_4, r_4 = r_4’, r_3 = r_3’, r_1 = r_1’, r_1 > r_2’ \} \{ r_2 < r_1 < r_4 < r_3, r_4 = r_4’, r_3 = r_3’, r_1 = r_1’, r_2 > r_1’ \}$

Note that we omit some atoms in $C_0$ and $C_1$ for readability: although they are not maximal (e.g. $C_0$ does not contain $r_2’ < r_1’ < r_4’ < r_3’$), they can be uniquely completed to maximal sets. Figure 2 (ignore the colored paths for now) visualises $C_0 C_1$ plus a bit more constraints. The black lines represent the evolution of the same register. The constraint $C_0$ describes the transition from moment 0 to 1, and $C_1$—from 1 to 2. This finite constraint sequence is satisfiable in $Q$ and in $N$. For example, the valuations can start with $\nu_0 = \{ r_4 \mapsto 6, r_3 \mapsto 5, r_2 \mapsto 4, r_1 \mapsto 3 \}$. But no valuations starting with $\nu_0(r_3) < 5$ can satisfy the sequence in $N$. Also, the constraint $C_0$ requires all registers in $R$ to differ, hence the sequence is not 0-satisfiable in $Q$ nor in $N$. Another example is given by the sequence $\{ \{r > r’\} \}^\omega$ with $R = \{ r \}$: it is satisfiable in $Q$ but not in $N$.

Satisfiability of constraint sequences in $Q$. The following result is glimpsed in several places (e.g. in [36, Appendix C]): a constraint sequence is satisfiable in $Q$ if it is consistent. This is a consequence of the following property which holds because $Q$ dense: for every constraint $C$ and $\nu \in Q^R$ such that $\nu \models C_{|R}$, there exists $\nu’ \in Q^R$ such that $\nu \cup \nu’ \models C$. Consistency can be checked by comparing every two consecutive constraints of the sequence. Thus it is not hard to show that consistent, hence satisfiable, constraint sequences in $Q$ are recognizable by deterministic parity automata (see Appendix A.1).

Theorem 1. There is a deterministic parity automaton of size exponential in $|R|$ that accepts exactly all constraint sequences satisfiable in $Q$. The same holds for 0-satisfiability.
Satisfiability of constraint sequences in $\mathbb{N}$. Fix $R$ and a constraint sequence $C_0 C_1 \ldots$ over $R$. A (decreasing) two-way chain is a finite or infinite sequence $(r_0, m_0) \triangleright_0 (r_1, m_1) \triangleright_1 \ldots \in (R \times \mathbb{N}) \cdot \{ (=, >) \}^{\omega}$ satisfying the following (note that $m_0$ can differ from 0).

- $m_{i+1} = m_i$, or $m_{i+1} = m_i + 1$ (time flows forward), or $m_{i+1} = m_i - 1$ (time goes backwards).
- If $m_{i+1} = m_i$ then $(r_i, r_{i+1}) \in C_{m_i}$.
- If $m_{i+1} = m_i + 1$ then $(r_i, r_{i+1}') \in C_{m_i}$.
- If $m_{i+1} = m_i - 1$ then $(r_{i+1}, r_i') \in C_{m_i - 1}$.

The depth of a chain is the number of $>$; when it is infinity, the chain is infinitely decreasing.

Figure 2 shows four two-way chains: e.g., the green-colored chain $(r_4, 2) > (r_3, 3) > (r_2, 2) > (r_1, 3) > (r_2, 3) \triangleright_3$ has depth 4. Similarly, we define one-way chains except that (a) they are either increasing (then $\triangleright \in \{ <, = \}$) or decreasing ($\triangleright \in \{ >, = \}$), and (b) time flows forward ($m_{i+1} = m_i + 1$) or stays ($m_{i+1} = m_i$). In Figure 2, the blue chain is one-way decreasing, the red chain is one-way increasing.

A stable chain is an infinite chain $(r_0, m) \triangleright_0 (r_1, m+1) \triangleright_1 (r_2, m+2) \triangleright_2 \ldots$ with all $\triangleright$ being the equality $=$; it can also be written as $(m, r_0 r_1 r_2 \ldots)$. Given a stable chain $\chi_r = (m, r_0 r_1 \ldots)$ and a chain $\chi_s = (s_0, n_0) \triangleright_0 (s_1, n_1) \triangleright_1 \ldots$, such that $n_i \geq m$ for all plausible $i$, the chain $\chi_r$ is non-strictly above $\chi_s$ if for all $n_i$ the constraint $C_{n_i}$ contains $r_{n_i - m} > s_{n_i}$ or $r_{n_i - m} = s_{n_i}$. A stable chain $(m, r_0 r_1 \ldots)$ is maximal if it is non-strictly above all other stable chains starting after $m$. In Figure 2, the yellow chain $(0, (r_2 r_3)^\omega)$ is stable, non-strictly above all other chains, and maximal. A trespassing chain is a chain that is below a maximal stable chain.

Lemma 2. A consistent constraint sequence is satisfiable in $\mathbb{N}$ iff

(A') it has no infinite-depth two-way chains; and

(B') $\exists b \in \mathbb{N}$: all trespassing two-way chains have depth $\leq b$ (i.e. they have bounded depth).

Proof idea. The left-to-right direction is trivial: if (A') is not satisfied, then one needs infinitely many values below the maximal initial value of a register to satisfy the sequence, which is impossible in $\mathbb{N}$. Likewise, if (B') is not satisfied, then one also needs infinitely many values below the value of a maximal stable chain, which is impossible. For the other direction, we show that if (A) and (B) hold, then one can construct a sequence of valuations $v_0 v_1 \ldots$ satisfying the constraint sequence, such that for all $r \in R$, $\nu_i(r)$ is the largest depth of a (decreasing) two-way chain starting in $r$ at moment $i$. The full proof is in Appendix A.2.

The previous lemma characterises satisfiability in terms of two-way chains, but our final goal is recognise it with an automaton. It is hard to design a one-way automaton tracing two-way chains, so we use a Ramsey argument to lift the previous lemma to one-way chains.

Lemma 3. A consistent constraint sequence is satisfiable in $\mathbb{N}$ iff

(A) it has no infinitely decreasing one-way chains and

(B) the trespassing one-way chains have a bounded depth.

Proof idea. We show that (A) and (B) implies (A') and (B'). (The other direction is simple). Consider $\neg A' \Rightarrow \neg A$. From an infinite (decreasing) two-way chain, we can always extract an infinite decreasing one-way chain, since two-way chains are infinite to the right and not to the left. Hence, for all moment $i$, there always exists a moment $j > i$ such that one register of the chain is smaller at step $j$ than a register of the chain at step $i$. We also prove that $\neg B' \Rightarrow \neg B$. Given a sequence of trespassing two-way chains of unbounded depth, we are able to construct a sequence of one-way chains of unbounded depth. This construction is more difficult than in the case $\neg A' \Rightarrow \neg A$. Indeed, even though there are by hypothesis deeper and deeper trespassing two-way chains, they may start at later and later moments in the constraint sequence and go to the left, and so one cannot just take an arbitrarily deep two-way chain...
and extract from it an arbitrarily deep one-way chain. However, we show, using a Ramsey argument, that it is still possible to extract arbitrarily deep one-way chains as the two-way chains are not completely independent. The full proof is in Appendix A.3.

The next lemma proved in Appendix A.4 refines the characterisation to \( 0 \)-satisfiability.

**Lemma 4.** A consistent constraint sequence is \( 0 \)-satisfiable in \( \mathbb{N} \) iff it satisfies conditions \( A \land B \) from Lemma 3, starts in \( C_0 \) s.t. \( C_0|_R = \{ r = s \mid r, s \in R \} \), and has no decreasing one-way chains of depth \( \geq 1 \) from \( (r, 0) \) for any \( r \).

We now state the main result about recognisability of satisfiable constraint sequences by max-automata [7]. These automata extend standard finite-alphabet automata with a finite set of counters \( c_1, \ldots, c_n \) which can be incremented, reset to 0, or updated by taking the maximal value of two counters, but they cannot be tested. The acceptance condition is given as a Boolean combination of conditions “counter \( c_i \) is bounded along the run”. Such a condition is satisfied by a run if there exists a bound \( b \in \mathbb{N} \) such that counter \( x_i \) has value at most \( b \) along the run. By using negation, conditions such as “\( x_i \) is unbounded along the run” can also be expressed. Deterministic max-automata are more expressive than \( \omega \)-regular automata. For instance, they can express the non-\( \omega \)-regular set of words \( w = a^n ba^{n^2}b \ldots \) such that \( n_i \leq b \) for all \( i \geq 0 \), for some \( b \in \mathbb{N} \) that can vary from word to word.

**Theorem 5.** For every \( R \), there is a deterministic max-automaton accepting exactly all constraint sequences satisfiable in \( \mathbb{N} \). The number of states is exponential in \( |R| \), and the number of counters is \( O(|R|^2) \). The same holds for \( 0 \)-satisfiability in \( \mathbb{N} \).

**Proof idea.** We design a deterministic max-automaton that checks conditions A and B of Lemma 3. Condition A, namely the absence of infinitely decreasing one-way chains, is checked as follows. We construct a nondeterministic Büchi automaton that guesses a chain and verifies that it is infinitely decreasing (“sees \( \infty \) infinitely often”). Determinising and complementing gives the sought deterministic parity automaton. Checking condition B (the absence of trespassing one-way chains of unbounded depth) is more involved. We design a master automaton that tracks every chain \( \chi \) that currently exhibits a stable behaviour. To every such chain \( \chi \), the master automaton assigns a tracer automaton whose task is to ensure the absence of unbounded-depth trespassing chains below \( \chi \). For that, it uses \( 2|R| \) counters and requires them to be bounded. The overall acceptance condition ensures that if the chain \( \chi \) is stable, then there are no trespassing chains below \( \chi \) of unbounded depth. Since the master automaton tracks every such potential chain, we are done. Finally, we take a product of all these automata, which preserves determinism. (See Appendix A.5.)

**Remark.** [36, Appendix C] shows that satisfiable constraint sequences in \( \mathbb{N} \) are characterised by nondeterministic \( \omega \)-B-automata [4], which are strictly more expressive than max-automata.

The next results will come handy for game-related problems.

**Lasso-shaped sequences (\( \omega \)-regularity).** An infinite sequence is lasso-shaped (or regular) if it is of the form \( w = uv^\omega \). Notice that the number of constraints over a finite number of registers \( R \) is finite. Thus, using the standard pumping argument, one can show that in regular sequences an unbounded chain eventually loops (the proof is in Appendix A.6):

**Lemma 6.** For every lasso-shaped consistent constraint sequence, it has trespassing one-way chains of unbounded depth iff it has trespassing one-way chains of infinite depth.

The above lemma together with Lemma 4 yields the following result:
Lemma 7. A lasso-shaped consistent constraint sequence is 0-satisfiable iff it has
- no infinite-depth decreasing one-way chains,
- no trespassing infinite-depth increasing one-way chains,
- no decreasing one-way chains of depth ≥ 1 from moment 0, and starts with \( C_0 \) s.t.
\[
C_{0|R} = \{ r = s \mid r, s \in R \}.
\]
The conditions of this lemma can be checked by an \( \omega \)-regular automaton (see Appendix A.7):

Theorem 8. For every \( R \), there is a deterministic parity automaton that accepts a constraint sequence iff it is consistent and satisfies the three conditions of Lemma 7; its number of states is exponential in \(|R|\) and its number of priorities is polynomial in \(|R|\).

Bounded sequences (data-assignment function). Fix a constraint sequence. Given a moment \( i \) and a register \( x \), a right two-way chain starting in \((x,i)\) \((r2w)\) is a two-way chain \((x,i) \triangleright (r_1,m_1) \triangleright (r_2,m_2) \triangleright \ldots\) such that \( m_j \geq i \) for all plausible \( j \). Note that \( r2w \) chains are two-way, meaning in particular that they can start and end in the same time moment \( i \).

We design a data-assignment function that maps satisfiable constraint sequence prefixes to register valuations satisfying it. The function assumes that the \( r2w \) chains in the prefixes are bounded. It also assumes every constraint \( C_i \) in the sequence satisfies the following: for all \( \nu \in D^R, \nu' \in D^R \) s.t. \( \nu \cup \nu' \models C_i \) : \( \{|r' \in R^i \mid \forall s \in R. \nu'(r') \neq \nu(s)\}| \leq 1 \) (assumption†). Intuitively: at most one new value can appear (but many disappear) during the step of the constraint (see also Appendix A.8). This assumption is used to simplify the proofs, yet it is satisfied by all constraint sequences induced by plays in Church games studied in the next section. A constraint sequence is meaningful if it is consistent, starts in \( C_0 \) with \( C_{0|R} = \{ r = s \mid r, s \in R \} \), and has no decreasing chains of depth ≥ 1 starting at moment 0.

Lemma 9 (data-assignment function). For every \( B \geq 0 \), there exists a data-assignment function \( f : (C_{0|R} \cup C^+) \to \mathbb{N}^R \) such that for every finite or infinite meaningful constraint sequence \( C_0C_1C_2\ldots \) satisfying assumption† and whose \( r2w \) chains are depth-bounded by \( B \), the register valuations \( f(C_{0|R})f(C_0)f(C_0C_1)\ldots \) satisfy the constraint sequence.

Proof idea. We define a special kind of \( x^{(m)}y^{(m)} \)-chains that help to estimate how many insertions between the values of \( x \) and \( y \) at moment \( m \) we can expect in future. As it turns out, without knowing the future, the distance between \( x \) and \( y \) has to be exponential in the maximal depth of \( x^{(m)}y^{(m)} \)-chains. We describe a data-assignment function that maintains such exponential distances (the proof is by induction). The function is surprisingly simple: if the constraint inserts a register \( x \) between two registers \( r \) and \( s \) with already assigned values \( d_r \) and \( d_s \), then set \( d_x = \lfloor d_r + d_s \rfloor \); and if the constraint puts a register \( x \) above all other registers, then set \( d_x = d_M + 2^m \) where \( d_M \) the largest value currently held in the registers and \( B \) is the given bound on the depth of \( r2w \) chains. Full proof is in Appendix A.9.

Church Synthesis Games

A Church synthesis game is a tuple \( G = (I, O, S) \), where \( I \) is an input alphabet, \( O \) is an output alphabet, and \( S \subseteq (I \cdot O)^\omega \) is a specification. Two players, Adam (the environment, who provides inputs) and Eve (the system, who controls outputs), interact. Their strategies are respectively represented as mappings \( \lambda_A : O^* \to I \) and \( \lambda_E : I^+ \to O \). Given \( \lambda_A \) and \( \lambda_E \), the outcome \( \lambda_A || \lambda_E \) is the infinite sequence \( i_0o_0i_1o_1\ldots \) such that for all \( j \geq 0 \) : \( i_j = \lambda_A(o_{0\ldots j-1}) \) and \( o_j = \lambda_E(i_{0\ldots j}) \). If \( \lambda_A || \lambda_E \in S \), the outcome is won by Eve, otherwise by Adam. Eve wins the game if she has a strategy \( \lambda_E \) such that for every Adam strategy \( \lambda_A \), the outcome \( \lambda_A || \lambda_E \)
is won by Eve. Solving a synthesis game amounts to finding whether Eve has a winning strategy. Synthesis games are parameterised by classes of alphabets and specifications. A game class is determined if every game in the class is either won by Eve or by Adam.

The class of synthesis games where $I$ and $O$ are finite and where $S$ is an $\omega$-regular language is known as Church games; they are decidable and determined. They also enjoy the finite-memoriness property: if Eve wins a game then there is an Eve winning strategy that can be represented as a finite-state machine.

We study synthesis games where the alphabets $I$ and $O$ are infinite and equipped with a linear order, and the specifications are described by deterministic register automata. Register automata. Fix a set of registers $R$. A test is a maximally consistent set of atoms of the form $\nu_{\infty} \in R$ for $\nu \in R$ and $\in \{=, <, >\}$. We may represent tests as conjunctions of atoms instead of sets. The symbol ‘$*$’ is used as a placeholder for incoming data. For example, for $R = \{r_1, r_2\}$, the expression $r_1 < *$ is not a test because it is not maximal, but $(r_1 < *) \land (\ast < r_2)$ is a test. We denote $\text{Tst}_R$ the set of all tests and just $\text{Tst}$ if $R$ is clear from the context. A register valuation $\nu \in \text{D}^R$ and data $d \in \text{D}$ satisfy a test $\text{tst} \in \text{Tst}$, written $(\nu, d) \models \text{tst}$, if all atoms of $\text{tst}$ get satisfied when we replace the placeholder $*$ by $d$ and every register $r \in R$ by $\nu(r)$. An assignment is a subset $\text{asgn} \subseteq R$. Given an assignment $\text{asgn}$, a data $d \in \text{D}$, and a valuation $\nu$, we define update$(\nu, d, \text{asgn})$ to be the valuation $\nu'$ s.t. $\forall r \in \text{asgn}: \nu'(r) = d$ and $\forall r \notin \text{asgn}: \nu'(r) = \nu(r)$.

A deterministic register automaton is a tuple $S = (Q, q_0, R, \delta, \alpha)$ where $Q = Q_A \sqcup Q_E$ is a set of states partitioned into Adam and Eve states, the state $q_0 \in Q_A$ is initial, $R$ is a set of registers, $\delta = \delta_A \sqcup \delta_E$ is a (total and deterministic) transition function $\delta_P : (Q_P \times \text{Tst} \rightarrow \text{Asgn} \times Q_P)$ for $P \in \{A, E\}$ and the other player $P'$, and $\alpha : Q \rightarrow \{1, ..., c\}$ is a priority function where $c$ is the priority index.

A configuration of $A$ is a pair $(q, \nu) \in Q \times \text{D}^R$, describing the state and register content; the initial configuration is $(q_0, 0^R)$. A run of $S$ on a word $w = d_0d_1 ... \in \text{D}^\omega$ is a sequence of configurations $\rho = (q_0, \nu_0)(q_1, \nu_1) ...$ starting in the initial configuration and such that for every $i \geq 0$: by letting $\text{tst}_i$ be a unique test for which $(\nu_i, d_i) \models \text{tst}_i$, we have $\delta(q_i, \text{tst}_i) = (\text{asgn}, q_{i+1})$ for some $\text{asgn}$, and $\nu_{i+1} = \text{update}(\nu_i, d_i, \text{asgn})$. Because the transition function $\delta$ is deterministic and total, every word induces a unique run in $S$. The run $\rho$ is accepting if the maximal priority visited infinitely often is even. A word is accepted by $S$ if it induces an accepting run. The language $L(S)$ of $S$ is the set of all words it accepts.

Church games on register automata. If the data domain is $(\mathbb{N}, \leq)$, Church games are undecidable. Indeed, if the two players pick data values, it is easy to simulate a two-counter machine, where one player provides the values of the counters and the other verifies that no cheating happens on the increments and decrements, using the fact that $c' = c + 1$ whenever there does not exist $d$ such that $c < d < c'$ (the formal proof can be found in Appendix B.1).

Theorem 10. Deciding the existence of a winning strategy for Eve in a Church game whose specification is a deterministic register automaton over $(\mathbb{N}, \leq)$ is undecidable.

Church games on one-sided register automata. In light of this undecidability result, we consider one-sided synthesis games, where Adam provides data but Eve reacts with labels from a finite alphabet (a similar restriction was studied in [20] for domain $(\mathbb{D}, =)$). Specifications are now given as a language $S \subseteq (\mathbb{D} \cdot \Sigma)^\omega$. Such games are still quite expressive, as they enable the synthesis of ‘relaying’ register transducers, which can only output data that is present in the specification automaton; we elaborate on this in Section 4.

A one-sided register automaton $S = (\Sigma, Q, q_0, R, \delta, \alpha)$ is a register automaton that additionally has a finite alphabet $\Sigma$ of Eve labels, and its transition function $\delta = \delta_A \sqcup \delta_E$ now
Theorem 1. For every Church game $G$ on a one-sided automaton $S$ over $\mathbb{N}$ or $\mathbb{Q}$:
1. Deciding if Eve wins $G$ is doable in time polynomial in $|Q|$ and exponential in $c$ and $|R|$.
2. The game is either won by Eve or otherwise by Adam.

Proof structure. We reduce the Church game $G$ to a finite-arena game $G_f$ called feasibility game. The states and transitions in $G_f$ are those of $S$, and a play is winning if it either satisfies the parity condition of $S$ or if the corresponding action word is not feasible.

In $\mathbb{Q}$, feasibility of action words can be checked by a deterministic parity automaton (Theorem 1). We then show that Eve wins the Church game $G$ iff she wins the finite-arena game $G_f$. The direction $\Rightarrow$ is easy, because Eve winning strategy $\lambda_E^f$ in $G_f$, which picks finite labels in $\Sigma$ depending on the history of transitions of $S$, can be used to construct Eve winning strategy $\lambda_E : Q^+ \rightarrow \Sigma$ in $G$ by simulating the automaton $S$. To prove the other direction, we assume that Adam has a winning strategy $\lambda_A^f$ in $G_f$, which picks tests depending on the history of transitions of $S$, then construct an Adam data strategy $\lambda_A : \Sigma^* \rightarrow Q$ that concretises these tests into data values. This data instantiation is easy because $Q$ is dense.

The case of $\mathbb{N}$ is treated similarly. However, checking feasibility of action words now requires a deterministic max-automaton (see page 7). From [8], we can deduce that games with a winning objective given as deterministic max-automata are decidable, yet the algorithm is involved, its complexity is high and does not yield finite-memory strategies that rely on picking transitions in $S$. Moreover, their determinacy is unknown. (For the same reasons we cannot rely on [6].) Therefore, we define quasi-feasible words, an $\omega$-regular subset of feasible words sufficient for our purpose, and correspondingly define an $\omega$-regular game $G_f^{reg}$ by strengthening the winning condition of $G_f$. We then show that the Church game $G$ and the finite-arena game $G_f^{reg}$ are equi-realisable. The hard direction is again to prove that if Eve wins in $G$, then she wins in $G_f^{reg}$. As for $\mathbb{Q}$, assuming that Adam wins in $G_f^{reg}$ with strategy $\lambda_A^f$, we construct Adam data strategy $\lambda_A : \Sigma^* \rightarrow \mathbb{Q}$, relying on the finite-memoriness of the strategy $\lambda_A^f$ and on the data-assignment function for constraint sequences from Lemma 9.

Remark 12. From the reduction of Church games to (quasi-)feasibility games, we get that if Eve wins a Church game $G$, then she has a winning strategy that simulates the run of the automaton $S$ and simply picks its transitions. In this sense, Eve’s strategy is ‘finite-memory’ as it can be expressed by a register automaton with outputs with a finite number of states.

Games on finite arenas. A two-player zero-sum finite-arena game (or just finite-arena game) is a tuple $G = (V_f, V_2, v_0, E, W)$ where $V_f$ and $V_2$ are disjoint finite sets of vertices controlled by Adam and Eve, $v_0 \in V_f$ is initial, $E \subseteq (V_f \times V_2) \cup (V_2 \times V_f)$ is a turn-based transition relation, and $W \subseteq (V_f \cup V_2)^\omega$ is a winning objective. An Eve strategy is a mapping $\lambda : (V_f \cdot V_2)^+ \rightarrow V_f$ such that $(v_2, \lambda(v_0 \cdots v_3)) \in E$ for all paths $v_0 \cdots v_3$ of $G$ starting in $v_0$ and ending in $v_2 \in V_2$. Adam strategies are defined similarly, by inverting the roles of $\exists$ and $\forall$. 
A play is a sequence of vertices starting in $v_0$ and satisfying the edge relation $E$. It is won by Eve if it belongs to $W$ (otherwise it is won by Adam). An infinite play $\pi = v_0v_1\ldots$ is compatible with an Eve strategy $\lambda$ when for all $i \geq 0$ s.t. $v_i \in V_2$: $v_{i+1} = \lambda(v_0\ldots v_i)$. An Eve strategy is winning if all infinite plays compatible with it are winning.

It is well-known that parity games can be solved in $n^c$ [23] (see also [13]), with $n$ the size of the game and $c$ the priority index.

**Feasibility games.** For the rest of this section, fix a one-sided register automaton $S = (\Sigma, Q, q_0, R, \delta, \alpha)$. With its Church game, we associate the following feasibility game, which is a finite-arena game $G_f = (V_f, V_3, v_0, E, W_f)$. Essentially, it memorises the transitions taken by the automaton $S$ during the play of Adam and Eve. It has $V_f = \{q_0\} \cup (\Sigma \times Q_A)$,

$V_3 = \text{Tst} \times \text{Asgn} \times Q_E$, $v_0 = q_0$, $E = E_0 \cup E_v \cup E_3$ where:

$E_0 = \{(v_0, (\text{tst}, \text{asgn}, u_0)) \mid \delta(v_0, \text{tst}) = (\text{asgn}, u_0)\}$,

$E_v = \{(\sigma, v), (\text{tst}, \text{asgn}, u) \mid \delta(v, \text{tst}) = (\text{asgn}, u)\}$, and

$E_3 = \{(\text{tst}, \text{asgn}, u), (\sigma, v) \mid \delta(u, \sigma) = v\}$.

Let Feasible$_{\geq 0}(R)$ denote the set of action words over $R$ feasible in $D$. We let:

$W_f = \{v_0(v_0, \text{asgn}, q_0)(a_0, v_1)\ldots (\text{tst}, \text{asgn}, q_0)\ldots \in \text{Feasible}_{\geq 0}(R) \Rightarrow v_0v_0v_1v_1\ldots = \alpha\}$

Later we will show that Eve wins the Church game $G$ iff she wins the feasibility game $G_f$.

**Action words and constraint sequences.** A constraint $C$ (cf Section 2) relates the values of the registers between the current moment and the next moment. A state constraint relates registers in the current moment only: it contains atoms over non-primed registers, so it has no atoms over primed registers. Note that both $C_{R'}$ and $\text{unprime}(C_{R'})$ are state constraints.

Every action word naturally induces a unique constraint sequence. For instance, for registers $R = \{r, s\}$, an action word starting with $\{\{r < s, s < s\}, \{s\}\}$ (test whether the current data $d$ is above the values of $r$ and $s$, store it in $s$) induces a constraint sequence starting with $\{r = s, r = r', s < s', r' < s'\}$ (the atom $r = s$ is due to all registers being equal initially). This is formalised in the next lemma, which is notation-heavy but says a simple thing: given an action word, we can construct, on the fly, a constraint sequence that is $0$-satisfiable iff the action word is feasible. For technical reasons, we need a new register $r_d$ to remember the last Adam data. The proof is direct and can be found in Appendix B.2.

**Lemma 13.** Let $R$ be a set of registers, $R_d = R \cup \{r_d\}$, and $D \in \{\mathbb{N}, \mathbb{Q}\}$. There exists a mapping $\text{constr} : \Pi \times \text{Tst} \times \text{Asgn} \rightarrow C$ from state constraints $\Pi$ over $R_d$ and tests-assignments over $R$ to constraints $C$ over $R_d$, such that for all action words $a_0a_1a_2\ldots \in (\text{Tst} \times \text{Asgn})^\omega$, $a_0a_1a_2\ldots$ is feasible iff $C_0C_1C_2\ldots$ is $0$-satisfiable, where $\forall i \geq 0$: $C_i = \text{constr}(\pi_i, a_i)$, $\pi_{i+1} = \text{unprime}(C_{i|R_d'}), \pi_0 = \{r = s \mid r, s \in R_d\}$.

**Expressing the winning condition of $G_f$ by deterministic automata.** By converting an action word to a constraint sequence and then testing its satisfiability, we can test whether the action word is feasible. This allows us to express the winning condition $W_f$ as a deterministic parity automaton for $D = \mathbb{Q}$ and as a deterministic max-automaton for $D = \mathbb{N}$. As a consequence of Theorem 1 (resp. 5), we get (see full proof in Appendix B.3):

**Lemma 14.** $W_f$ is definable by a deterministic parity automaton if $D = \mathbb{Q}$ and a deterministic max-automaton if $D = \mathbb{N}$. Moreover, these automata are polynomial in $|Q|$ and exponential in $|R|$, and for $D = \mathbb{Q}$, the index of the priority function is linear in $c$.

**Solving synthesis games on $(\mathbb{Q}, \leq)$**

We outline the proof of Theorem 11 for $(\mathbb{Q}, \leq)$; the full proof can be found in Appendix B.4.
We now outline the proof of Theorem 11 for \( \omega \). As \( \omega \)-regular games are determined, Adam has a winning strategy \( \lambda_A^f \) in \( G_f \). It induces a strategy \( \lambda_A \) for Adam in \( G \): when the test is an equality, pick the corresponding data, and when it is of the form \( r < * < r' \), take some rational number strictly in the interval. Then, each play consistent with this strategy in \( G \) corresponds to a unique run in \( S \), which is also a play in \( G_f \). As \( \lambda_A^f \) is winning, such run is accepting, so \( \lambda_A \) is winning: Eve does not win \( G \).

Since the feasibility game \( G_f \) is of size polynomial in \( |Q| \) and exponential in \( |R| \), and has a number of priorities linear in \( c \), we obtain item \((b)\) of the theorem. Item \((c)\) (determinacy) and Remark 12 are then a consequence of the finite-memory determinacy of \( \omega \)-regular games.

**Solving synthesis games on \((\mathbb{N}, \leq)\)**

We now outline the proof of Theorem 11 for \((\mathbb{N}, \leq)\); the full proof is in Appendix B.5.

**Using \( \omega \)-regular game \( G_{f}^{\text{reg}} \) instead of \( G_f \).** \( W_f \) is not \( \omega \)-regular, and the known results over deterministic max-automata do not suffice to obtain determinacy nor finite-memoriness, which will both prove useful for the transducer synthesis problem (cf Section 4).

We thus define an \( \omega \)-regular subset \( W_f^{\text{reg}} \subseteq W_f \) which is equi-realisable to \( W_f \). Let \( \text{QFeasible}_R \) be the set of quasi-feasible action words over \( R \), defined as the set of words \( \pi \) such that its induced constraint sequence (through the mapping \( \text{constr} \) of Lemma 13) starts with \( C_0 \), has no infinite-depth decreasing one-way chain nor trespassing increasing one-way chain, and no decreasing one-way chain of depth \( \geq 1 \) from moment \( 0 \).

We then let:

\[
W_f^{\text{reg}} = \{ v_0(tst_0, \text{asgn}_0, u_0)(s_0, v_1) \ldots | (tst_0, \text{asgn}_0) \ldots \in \text{QFeasible}_R \Rightarrow v_0u_0v_1u_1 \ldots \models \alpha \}.
\]

From Theorem 8, we can build a deterministic parity automaton with a number of states exponential in \( |R| \) and polynomial in \( |Q| \) and a priority index polynomial in \( c \) recognising \( W_f^{\text{reg}} \). Let \( G_f^{\text{reg}} \) be the finite-arena game with the same arena as \( G_f \), with winning condition \( W_f^{\text{reg}} \). We now show that the Church game \( G \) reduces to \( G_f^{\text{reg}} \) (full proof in Appendix B.5).

**Proposition 15.** Eve has a winning strategy in \( G \) iff she has a winning strategy in \( G_f^{\text{reg}} \).

**Proof idea.** If Eve has a winning strategy in \( G_f^{\text{reg}} \), then, since \( \text{Feasible}_R \subseteq \text{QFeasible}_R \), we have that \( W_f^{\text{reg}} \subseteq W_f \), so it is also winning in \( G_f \). Now, the argument for \( Q \) applies again for \( \mathbb{N} \); as Eve has more information in \( G \), if she wins in \( G_f \), she wins in \( G \).

The converse implication is harder; we show it by contraposition. Assume Eve does not have a winning strategy in \( G_f^{\text{reg}} \). As \( \omega \)-regular games are finite-memory determined, Adam has a finite-memory winning strategy \( \lambda_A^f \) in \( G_f^{\text{reg}} \). It is not clear a priori that such strategy can be instantiated to a winning data strategy in \( G \). However, we show that for finite-memory strategies, the depth of right two-way chains is uniformly bounded, which by Lemma 9 allows us to instantiate the tests with concrete data:

**Lemma 16.** There is a number \( b \geq 0 \) that bounds the depths of all \( r2w \) chains coming from \( \lambda_A^f \); for all constraint sequences resulting from playing with \( \lambda_A^f \), for all \( x \in R \), for all \( i \geq 0 \), we have that for all \( r2wch \) from \( (x, i) \), \( \text{depth}(r2wch) \leq b \).

**Proof idea of the lemma.** Fix a moment \( i \) and a register \( x \). After the moment \( i \), only a bounded number of values can be inserted below the value of register \( x \) at moment \( i \). Similarly, if we fix two registers at moment \( i \), there can only be a bounded number of insertions between the values of \( x \) and \( y \) at moment \( i \). Indeed, by finite-memoriness of Adam strategy, once the number of such insertions is larger than the memory of Adam, Eve can repeat her actions.
to force an infinite number of such insertions, leading to a play with an unfeasible action sequence and hence won by Eve. This intuition is captured by r2w chains defined in Section 2. We prove the lemma by contradiction, by constructing a play consistent with \( \lambda_A \) which induces an unsatisfiable constraint sequence and therefore is losing for Adam. Assume that the constraint sequences induced by the plays with \( \lambda_A \) have unbounded-depth 2w chains. By Ramsey argument from Lemma 2, the constraint sequences have unbounded-depth 1w chains. Along those chains, as \( \lambda_A \) is finite-memory, there is a repeating configuration with the same constraints and states, and where the chain decrements or increments at least once and goes through the same registers. Thus, we can define a strategy \( \lambda_E \) of Eve which loops there forever. This induces an infinite chain. If it is decreasing, the corresponding play is not feasible, and is thus losing for Adam. If it is increasing, recall that this chain is actually a part of a r2w chain. By gluing them together, we get a r2w chain of infinite depth, which is not feasible either (recall that r2w chains start and end at the same point of time), so it is again losing for Adam. In both cases, this contradicts the assumption that \( \lambda_A \) is winning. ◀

Now, thanks to this uniform bound \( b \) and Lemma 9, we can construct \( \lambda_N^A \) from \( \lambda_A \) by translating the currently played action-word prefix \((\text{tst}_0, \text{asgn}_0)\ldots(\text{tst}_m, \text{asgn}_m)\) into a constraint-sequence prefix and applying the data-assignment function to it. By construction, for each play in \( G \) consistent with \( \lambda_N^A \), the corresponding run in \( S \) is a play consistent with \( \lambda_A \) in \( G_{\text{reg}^f} \). As \( \lambda_A \) is winning, such run is not accepting, i.e. the play is winning for Adam in \( G \). Therefore, \( \lambda_N^A \) is a winning Adam’s strategy in \( G \), meaning that Eve loses \( G \). ◀

Since \( G_{\text{reg}^f} \) is of size polynomial in \(|Q|\) and exponential in \(|R|\), Theorem 11 follows.

## 4 Application to Transducer Synthesis

We now apply the above results to the transducer synthesis problem for specifications defined by input-driven register automata [17], i.e. two-sided automata where the output data is restricted to be the content of some register. Formal definitions of input-driven register automata and of register transducers are omitted as they are straightforward generalisations to the ordered case. Given a register automaton specification \( S \), the transducer synthesis problem asks whether there exists a register transducer \( T \) such that \( L(T) \subseteq L(S) \). A priori, \( T \) and \( S \) can have different sets of registers, but we show that it suffices to consider implementations that are subautomata of \( S \), a result reminiscent of [17, Proposition 5]. Definitions and full proof of the theorem can be found in Appendix B.6.

**Theorem 17.** For specifications defined by deterministic input-driven output register automata over data domains \( Q, N \) and \( N \), the register transducer synthesis problem can be solved in time polynomial in \(|Q|\) and exponential in \( c \) and \(|R|\).

**Proof idea.** The transducer synthesis problem reduces to solving a one-sided Church game \( G \). Indeed, output registers can be treated as finite labels, up to remembering equality constraints between registers in the states (this is exponential in \(|R|\), but the exponentials do not stack). Moreover, we know by Proposition 15 that \( G \) itself reduces to \( G_{\text{reg}^f} \). If Eve wins \( G_{\text{reg}^f} \), she has a finite-memory winning strategy, which corresponds to a register transducer implementation of \( S \) which behaves like a subautomaton of \( S \).

## 5 Conclusion

In this paper, our main result states that 1-sided Church games for specifications given as deterministic register automata over \((N, \leq)\) are decidable, in \textsc{ExpTime}. Moreover, we show
that those games are determined. 1-sided Church games are motivated by register transducer synthesis, and the above result provides an \textit{ExpTime} algorithm for this problem. As a future direction, it seems important to consider more expressive specification languages. Indeed, deterministic register automata are known to be strictly less expressive than nondeterministic or universal register automata. Such extensions are known to yield undecidability when used as specification formalisms in 1-sided Church games, already in the case of data equality only [17]. In [17, 28], a parameterized version of 1-sided Church games is shown to be decidable for universal register automata specifications. The parameter is a positive integer \( k \) and the goal is to decide whether there exists a strategy which can be implemented as a transducer with \( k \) registers. We plan to extend this result to linear orders. Universal register automata, thanks to their universal transitions, are better suited to specify properties of reactive systems. As an example, they can easily model properties such as “every request of client \( i \) is eventually granted”, for every client id \( i \in \mathbb{N} \). Such properties are not expressible by deterministic nor nondeterministic register automata. On the data part, while equality tests are sufficient for such properties, having a linear order could allow us to express more complex but natural properties, e.g. involving priorities between clients.

An important future direction is to consider logical formalisms instead of automata to describe specifications in a more declarative and high-level manner. Data-word first-order logics [5, 35] have been studied with respect to the satisfiability problem but when used as specification languages for synthesis, only few results are known. For slightly different contexts, see for example [3] for parameterized synthesis and [20] for games with temporal specifications and data.
References

1. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Piotr Hofman, Richard Mayr, K. Narayan Kumar, and Patrick Totzke. Infinite-state energy games. In Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS ’14, Vienna, Austria, July 14 - 18, 2014, pages 7:1–7:10, 2014.

2. Parosh Aziz Abdulla, Ahmed Bouajjani, and Julien d’Orso. Deciding monotonic games. In International Workshop on Computer Science Logic, pages 1–14. Springer, 2003.

3. Béatrice Bérard, Benedikt Bollig, Mathieu Lehaut, and Nathalie Sznajder. Parameterized synthesis for fragments of first-order logic over data words. In FOSSACS, volume 12077 of Lecture Notes in Computer Science, pages 97–118. Springer, 2020.

4. M. Bojańczyk and T. Colcombet. Bounds in $\omega$-regularity. In Proc. 21st IEEE Symp. on Logic in Computer Science, pages 285–296, 2006.

5. M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In Proc. 21st IEEE Symp. on Logic in Computer Science, pages 7–16, 2006.

6. Mikolaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, Computer Science Logic, pages 41–55, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

7. Mikolaj Bojańczyk. Weak MSO with the unbounding quantifier. Theory of Computing Systems, 48(3):554–576, 2011.

8. Mikolaj Bojańczyk. Weak MSO+U with path quantifiers over infinite trees. In Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II, pages 38–49, 2014.

9. A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In FCT, pages 1–22, 2007.

10. A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. Theoretical Computer Science, 295:86–106, 2003.

11. A.-J. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with unboundedness and regular conditions. In Proc. 23rd Conf. on Foundations of Software Technology and Theoretical Computer Science, volume 2914 of Lecture Notes in Computer Science, pages 88–99. Springer, 2003.

12. J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. Trans. AMS, 138:295–311, 1969.

13. C.S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In Proc. 49th ACM Symp. on Theory of Computing, pages 252–263, 2017.

14. S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

15. G. Delzanno, A. Sangnier, and R. Traverso. Parameterized verification of broadcast networks of register automata. In P. A. Abdulla and I. Potapov, editors, Reachability Problems, pages 109–121, Berlin, Heidelberg, 2013. Springer.

16. R. Ehlers, S. Seshia, and H. Kress-Gazit. Synthesis with identifiers. In Proc. 15th Int. Conf. on Verification, Model Checking, and Abstract Interpretation, volume 8318 of Lecture Notes in Computer Science, pages 415–433. Springer, 2014.

17. L. Exibard, E. Filiot, and P-A. Reynier. Synthesis of data word transducers. In Proc. 30th Int. Conf. on Concurrency Theory, 2019.

18. Rachel Faran and Orna Kupferman. On synthesis of specifications with arithmetic. In Alexander Chatzigeorgiou, Riccardo Dondi, Herodotos Herodotou, Christos Kapoutsis, Yannis Manolopoulos, George A. Papadopoulos, and Florian Sikora, editors, SOFSEM 2020: Theory and Practice of Computer Science, pages 161–173, Cham, 2020. Springer International Publishing.
Azadeh Farzan and Zachary Kincaid. Strategy synthesis for linear arithmetic games. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–30, 2017.

Diego Figueira, Anirban Majumdar, and M. Praveen. Playing with repetitions in data words using energy games. *Log. Methods Comput. Sci.*, 16(3), 2020. URL: https://lmcs.episciences.org/6614.

B. Finkbeiner, F. Klein, R. Piskac, and M. Santolucito. Temporal stream logic: Synthesis beyond the bools. In *Proc. 31st Int. Conf. on Computer Aided Verification*, 2019.

Stefan Göller, Richard Mayr, and Anthony Widjaja To. On the computational complexity of verifying one-counter processes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science*, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA, pages 235–244, 2009.

E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of Lecture Notes in Computer Science. Springer, 2002.

R. Hojati, D.L. Dill, and R.K. Brayton. Verifying linear temporal properties of data insensitive controllers using finite instantiations. In *Hardware Description Languages and their Applications*, pages 60–73. Springer, 1997.

M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

A. Khalimov, B. Maderbacher, and R. Bloem. Bounded synthesis of register transducers. In *16th Int. Symp. on Automated Technology for Verification and Analysis*, volume 11138 of Lecture Notes in Computer Science, pages 494–510. Springer, 2018.

Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In *30th International Conference on Concurrency Theory (CONCUR 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019*, August 27-30, 2019, Amsterdam, the Netherlands, volume 140 of LIPIcs, pages 25:1–25:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2019.25.

Bartek Klin and Mateusz Łełyk. Scalar and Vectorial mu-calculus with Atoms. *Logical Methods in Computer Science*, Volume 15, Issue 4, Oct 2019. URL: https://lmcs.episciences.org/5877, doi:10.23638/LMCS-15(4:5)2019.

Paul Krogmeier, Umang Mathur, Adithya Murali, P. Madhusudan, and Mahesh Viswanathan. Decidable synthesis of programs with uninterpreted functions. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 634–657, Cham, 2020. Springer International Publishing.

R. Lazić and D. Nowak. A unifying approach to data-independence. In *Proc. 11th Int. Conf. on Concurrency Theory*, pages 581–596. Springer Berlin Heidelberg, 2000.

M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1 edition, 1967.

N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.

M.O. Rabin. Automata on infinite objects and Church’s problem. *Amer. Mathematical Society*, 1972.

Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. *Log. Methods Comput. Sci.*, 8(1), 2012.

Luc Segoufin and Szymon Torunczyk. Automata-based verification over linearly ordered data domains. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

Olivier Serre. Parity games played on transition graphs of one-counter processes. In *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS*.
38 V. Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT ’09*, pages 1–13, 2009.

39 I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Proc. 20th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2000.

40 P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 184–192, 1986.
A.1 Proof of Theorem 1

To establish the result formally, we first show the following lemma.

Lemma 18. Let $R$ be a set of registers and $\mathcal{D} = \mathbb{Q}$. A constraint sequence $C_0C_1\ldots$ is satisfiable iff it is consistent. It is 0-satisfiable iff it is consistent and $C_{0|R} = \{r_1 = r_2 \mid r_1, r_2 \in R\}$.

Proof. Direction $\Rightarrow$ is simple for both claims, so we only prove direction $\Leftarrow$.

Consider the first claim, direction $\Leftarrow$. Assume the sequence is consistent. We construct $\nu_0\nu_1\ldots \in (Q^R)^\omega$ such that $\nu_i \cup \nu'_{i+1} \models C_i$ for all $i$. The construction proceeds step-by-step and relies on the following fact (†): for every constraint $C$ and $\nu \in Q^R$ such that $\nu \models C$, there exists $\nu' \in Q^R$ such that $\nu \cup \nu' \models C$. Then define $\nu_0, \nu_1 \ldots$ as follows: start with an arbitrary $\nu_0$ satisfying $\nu_0 \models C_{0|R}$. Given $\nu_i \models C_{i|R}$, let $\nu_{i+1}$ be any valuation in $Q^R$ that satisfies $\nu_i \cup \nu'_{i+1} \models C_i$ (it exists by (†)). Since $\nu_{i+1} \models C_{i+1|R}$, and unprime$(C_{i|R}) = C_{i+1|R}$ by consistency, we have $\nu_{i+1} \models C_{i+1|R}$, and we can apply the argument again.

We are left to prove the fact (†). The constraint $C$ completely specifies the order on $R \cup R'$, while $\nu$ fixes the values for $R$, and $\nu \models C_{1|R}$. Hence we can uniquely order registers $R'$ and the values $\{\nu(r) \mid r \in R\}$ on $R$ on the $Q$-line. Since $Q$ is dense, it is always possible to choose the values for $R'$ that respect this order; we leave out the details.

Consider the second claim, direction $\Leftarrow$. Since $C_0C_1\ldots$ is consistent, then by the first claim, it is satisfiable, hence it has a witnessing valuation $\nu_0\nu_1\ldots$. The constraint $C_0$ requires all registers in $R$ to start with the same value, so define $d = \nu_0(r)$ for arbitrary $r \in R$. Let $\nu'_{0}\nu'_{1}\ldots$ be the valuations decreased by $d$: $\nu'_i(r) = \nu_i(r) - d$ for every $r \in R$ and $i \geq 0$. The new valuations satisfy the constraint sequence because the constraints in $Q$ are invariant under the shift (follows from the fact: if $r_1 < r_2$ holds for some $\nu \in \mathcal{D}^R$, then it holds for any $\nu - d$ where $d \in \mathcal{D}$). The equality $\nu'_0 = 0^R$ means that the constraint sequence is 0-satisfiable.

We now prove Theorem 1.

Proof of Theorem 1. We describe the parity automaton. Its alphabet consists of all constraints. By Lemma 18, for satisfiability, it suffices to construct the automaton that checks consistency, namely that every two adjacent constraints $C_1C_2$ in the input word satisfy the condition unprime$(C_{1|R'}) = C_{2|R}$. The construction is straightforward; we only sketch it. The automaton memorises the atoms $C_{1|R'}$ of the last constraint $C_1$ into its state, and on reading the next constraint $C_2$ the automaton checks that unprime$(C_{1|R'}) = C_{2|R}$. If this holds, the automaton transits into the state that remembers $C_{2|R'}$; if the check fails, the automaton goes into the rejecting sink state. And so on. The number of states is exponential in $|R|$, the parity index is 1. The automaton for checking 0-satisfiability additionally checks that $C_{0|R} = \{r = s \mid r, s \in R\}$.

A.2 Proof of Lemma 2

Proof. Direction $\Rightarrow$: Suppose a constraint sequence $C_0C_1\ldots$ is satisfiable by some valuations $\nu_0\nu_1\ldots$. Assume $\neg A'$: there is an infinite decreasing two-way chain $\chi = (r_0, m_0)(r_1, m_1)\ldots$. Let $\nu_{m_0}(r_0) = d^*$ be the data value at the start of the chain. Each decrease $(r_i, m_i) > (r_{i+1}, m_{i+1})$ in the chain $\chi$ requires the data to decrease as well: $\nu_i(r_i) > \nu_{i+1}(r_{i+1})$. Hence there must be an infinite number of data values between $d^*$ and 0, which is impossible in $\mathbb{N}$. Hence $A'$
must hold. Now assume \( \neg B' \): there is a sequence of two-way trespassing chains of unbounded depth. By definition of trespassing, the constraint sequence has a maximal stable chain. Let \( d^* \) be the value of the registers in the maximal stable chain. All trespassing chains lay non-strictly below the maximal stable chain, therefore the values of their registers are bounded by \( d^* \). Hence the depths of such chains are bounded by \( d^* \), contradicting the assumption \( \neg B' \), so \( B' \) holds.

Direction \( \Leftarrow \). Given a consistent constraint sequence \( C_0C_1\ldots \) satisfying \( A' \) and \( B' \), we construct a sequence of register valuations \( \nu_0\nu_1\ldots \) such that \( \nu_i \cup \nu_{i+1} \models C_i \) for all \( i \geq 0 \) (recall that \( \nu' = \{ r' \mapsto \nu(r) \mid r \in R \} \)). For a register \( r \) and moment \( i \in \mathbb{N} \), let \( d(r,i) \) be the largest depth of two-way chains from \( (r,i) \); the depth \( d(r,i) \) can be 0 but not \( \infty \), by assumption \( A' \).

Then, for every \( r \in R \) and \( i \in \mathbb{N} \), set \( \nu_i(r) = d(r,i) \).

We now prove that for all \( i \), the satisfaction \( \nu_i \cup \nu_{i+1} \models C_i \) holds, i.e. all atoms of \( C_i \) are satisfied. Pick an arbitrary atom \( t_1 \gg t_2 \) of \( C_i \), where \( t_1,t_2 \in R \cup R' \). Define \( m_{t_1} = i+1 \) if \( t_1 \) is a primed register, else \( m_{t_1} = i \); similarly define \( m_{t_2} \). There are two cases.

- \( t_1 \gg t_2 \) is \( t_1 = t_2 \). Then the deepest chains from \( (t_1,m_{t_1}) \) and \( (t_2,m_{t_2}) \) have the same depth, \( d(t_1,m_{t_1}) = d(t_2,m_{t_2}) \), and hence \( \nu_i \cup \nu_{i+1} \) satisfies the atom.
- \( t_1 \gg t_2 \) is \( t_1 > t_2 \). Then, any chain \( (t_2,m_{t_2}) \ldots \) from \( (t_2,m_{t_2}) \) can be prefixed by \( (t_1,m_{t_1}) \) to create the deeper chain \( (t_1,m_{t_1}) > (t_2,m_{t_2}) \ldots \). Hence \( d(t_1,m_{t_1}) > d(t_2,m_{t_2}) \), therefore \( \nu_i \cup \nu_{i+1} \) satisfies the atom.

This concludes the proof.

### A.3 Proof of Lemma 3

**Proof.** We show that the conditions \( A \land B \) hold iff the conditions \( A' \land B' \) from Lemma 2 hold. The directions \( \neg A \Rightarrow \neg A' \) and \( \neg B \Rightarrow \neg B' \) follow from the definitions of chains.

Direction \( \neg A' \Rightarrow \neg A \). Given an infinite two-way chain \( \chi = (r_a,i)\ldots \), we construct an infinite descending one-way chain \( \chi' \). The construction is illustrated in Figure 3. Our one-way chain \( \chi' \) starts in \( (r_a,i) \). The area on the left from \( i \)-timeline contains \( i \cdot |R| \) points, but \( \chi \) has an infinite depth hence at some point it must go to the right from \( r_a \). Let \( r_b \) be the smallest register visited at moment \( i \) by \( \chi \); we first assume that \( r_b \) is different from \( r_a \) (the other case is later). Let \( \chi \) go \( (r_b,i) \triangleright (r',i+1) \). We append this to \( \chi' \) and get \( \chi' = (r_a,i) > (r_b,i) \triangleright (r',i+1) \). If \( r_a \) and \( r_b \) were actually the same, so the chain \( \chi \) moved \( (r_a,i) \triangleright (r',i+1) \), then we would append only \( (r_a,i) \triangleright (r',i+1) \). By repeating the argument from the point \( (r',i+1) \), we construct the infinite descending one-way chain \( \chi' \). Hence \( \neg A \) holds.

Direction \( \neg B' \Rightarrow \neg B \). Given a sequence of trespassing two-way chains of unbounded depth, [...]

![Figure 3](image-url) Proving the direction \( \neg A' \Rightarrow \neg A \) in Lemma 3. The two-way chain is in black, the constructed one-way chain is in blue.
depth, we need to create a sequence of trespassing one-way chains of unbounded depth. We extract a witnessing one-way chain of a required depth from a sufficiently deep two-way chain. To this end, we represent the two-way chain as a clique with colored edges, and whose one-colored subcliques represent all one-way chains. We then use the Ramsey theorem that says a monochromatic subclique of a required size always exists if a clique is large enough. From the monochromatic subclique we extract the sought one-way chain.

The Ramsey theorem is about clique graphs with colored edges. For the number $n \in \mathbb{N}$ of vertices, let $K_n$ denote the clique graph and $E_{K_n}$ — its edges, and let $\text{color}: E_{K_n} \to \{1, \ldots, \#c\}$ be the edge-coloring function, where $\#c$ is the number of edge colors in the clique. A clique is monochromatic if all its edges have the same color ($\#c = 1$). The Ramsey theorem says:

Fix the number $\#c$ of edge colors. $(\forall n)(\exists l)(\forall \text{color}: E_{K_l} \to \{1, \ldots, \#c\})$: there exists a monochromatic subclique of $K_l$ with $n$ vertices. The number $l$ is called Ramsey number for $(\#c, n)$.

I.e., for any given $n$, there is a sufficiently large size $l$ such that any colored clique of this size contains a monochromatic subclique of size $n$. We will only use $\#c = 3$.

Given a sequence of two-way chains of unbounded depth, we show how to build a sequence of one-way chains of unbounded depth. Suppose we want to build a one-way chain of depth $n$, and let $l$ be Ramsey number for $(3, n)$. Because the two-way chains from the sequence have unbounded depth, there is a two-way chain $\chi$ of depth $l$. From it we construct the following colored clique (the construction is illustrated in Figure 4).

- Remove stuttering elements from $\chi$: whenever $(r_i, m_i) = (r_{i+1}, m_{i+1})$ appears in $\chi$, remove $(r_{i+1}, m_{i+1})$. We repeat this until no stuttering elements appear. Let $\chi_\triangleright = (r_1, m_1) > \cdots > (r_l, m_l)$ be the resulting sequence; it is strictly decreasing, and contains $l$ pairs (the same as the depth of the original $\chi$). Note the following property $(\triangleright)$: for every
not necessarily adjacent \((r_i, m_i) > (r_j, m_j)\), there is a one-way chain \((r_i, m_i) \ldots (r_j, m_j)\); it is decreasing if \(m_i < m_j\), and increasing otherwise; its depth is at least \(1\).

The elements \((r, m)\) of \(\chi>\) serve as the vertices of the colored clique. The edge-coloring function is: for every \((r_a, m_a) > (r_b, m_b)\) in \(\chi>\), let \(\text{color}((r_a, m_a), (r_b, m_b))\) be \(\nearrow\) if \(m_a < m_b\), \(\searrow\) if \(m_a > m_b\), \(\downarrow\) if \(m_a = m_b\). Figure 4b gives an example.

By applying the Ramsey theorem, we get a monochromatic subclique of size \(n\) with vertices \(V \subseteq \{(r_1, m_1), \ldots, (r_1, m_1)\}\). Its color cannot be \(\downarrow\) when \(n > |R|\), because a time line has maximum \(|R|\) points. Suppose the subclique color is \(\nearrow\) (the case of \(\searrow\) is similar). We build the increasing sequence \(\chi^* = (r_1^*, m_1^*) < \cdots < (r_n^*, m_n^*)\), where \(m_i^* < m_{i+1}^*\) and \((r_i^*, m_i^*) \in V\), for every plausible \(i\). The sequence \(\chi^*\) may not satisfy the definition of one-way chains, because the removal of stuttering elements that performed at the beginning can cause time jumps \(m_{i+1} > m_i + 1\). But it is easy—relying on the property (\(\downarrow\))—to construct the one-way chain \(\chi^{**}\) of depth \(n\) from \(\chi^*\) by inserting the necessary elements between \((r_i, m_i)\) and \((r_{i+1}, m_{i+1})\). Finally, when the subclique has color \(\searrow\), the resulting chain is decreasing.

Thus, for every given \(n\), we constructed either a decreasing or increasing trespassing one-way chain of depth \(n\)—in other words, a sequence of such chains of unbounded depth. Hence \(\neg B\) holds, which concludes the proof of direction \(\neg B' \Rightarrow \neg B\).

\section*{A.4 Proof of Lemma 4}

Proof. Direction \(\Rightarrow\). The first two items follow from the definition of satisfiability and Lemma 3. Consider the last item: suppose there is such a chain. Then, at the moment when the chain strictly decreases and goes to some register \(s\), the register \(s\) would need to have a value below \(0\), which is impossible in \(\mathbb{N}\).

Direction \(\Leftarrow\). Since the conditions \(A \land B\) hold, the sequence is satisfiable, hence it also satisfies the conditions \(A' \land B'\) from Lemma 2. In the proof of Lemma 2, we showed that in this case the following valuations \(v_0 v_1\ldots\) satisfy the sequence: for every \(r \in R\) and moment \(i \in \mathbb{N}\), set \(v_i(r)\) (the value of \(r\) at moment \(i\)) to the largest depth of the two-way chains starting in \((r, i)\). We construct \(v_0 v_1\ldots\) as above, and get a witness of satisfaction of our constraint sequence. But note that at moment \(0\), \(v_0 = 0^R\), by the last item. Hence the constraint sequence is \(0\)-satisfiable.

\section*{A.5 Proof of Theorem 5}

Proof. We describe a max-automaton \(A\) that accepts a constraint sequence iff it is consistent and has no infinitely decreasing 1w chains and no trespassing 1w chains of unbounded depth. By Lemma 3, such a sequence is satisfiable.

The automaton has three components \(A = A_c \land A_{\infty} \land A_u\).

\(A_c\). The parity automaton \(A_c\) checks consistency, i.e. that \(\forall i: \text{unprime}(C_i|_R) = (C_{i+1})|_R\).

\(A_{\infty}\). The parity automaton \(A_{\infty}\) ensures there are no infinitely decreasing 1w chains. First, we construct its negation, an automaton that accepts a constraint sequence iff it has such a chain. Intuitively, the automaton guesses such a chain and then verifies that the guess is correct. It loops in the initial state \(q_0\) until it nondeterministically decides that now is the starting moment of the chain and guesses the first register \(r_0\) of the chain, and transits into the next state while memorising \(r_0\). When the automaton is in a state with \(r\) and reads a constraint \(C\), it guesses the next register \(r_n\), verifies that \((r_n > r) \in C\) or \((r_n = r) \in C\), and transits into the state that remembers \(r_n\). The Büchi acceptance condition ensures that the automaton leaves the initial state and transits from some \(r\) to some \(r_n\) with \((r_n > r) \in C\) infinitely often. Determinising and complementing this automaton gives \(A_{\infty}\).
States of \( A \). Before describing \( A \), we define ‘levels’. Fix a constraint \( C \). A level \( l \subseteq R \) describes an equivalence class of registers wrt. \( C|_R \) or wrt. \( \text{unprime}(C|_{R'}) \). Thus, in the constraint \( C \) we distinguish two sets of levels: at the beginning of the step, called start levels, and at the end of the step, called end levels. A start level \( l \subseteq R \) disappears when \( C \) contains no atoms of the form \( r = s' \) for \( r \in l \) and \( s \in R \). An end level \( l \subseteq R \) is new if \( C \) contains no atoms of the form \( r = s' \) where \( r \in R \) and \( s \in l \). A start level \( l \) morphs into an end level \( l' \) if \( C \) contains an atom \( r = s' \) for some \( r \in l \) and \( s \in l' \). Figure 5 illustrates the definitions. Notice that there can be at most \( |R| \) start and \( |R| \) end levels. We now describe \( A \).

The max-automaton \( A \) ensures there are no unbounded-depth trespassing 1w chains. It relies on the team of \( |R| \) chain tracers \( \overrightarrow{T} = \{t_{r_1}, ..., t_{r_{|R|}}\} \). Each tracer \( t_r \) is equipped with a counter \( \text{idle}_r \) and a set \( C_{n_r} \) of \( 2|R| \) counters. The tracers are controlled by the master automaton via four commands \( \text{idle}, \text{start}, \text{move}, \text{reset} \). We first describe the master automaton and then the tracers.

Master. States of \( A \) are of the form \( (\overrightarrow{t_r}, \overrightarrow{q}) \), where the mapping \( \overrightarrow{t_r} \) maps levels to tracers (a tracer will track chains below a level), \( \overrightarrow{q} = (q_{t_{r_1}}, ..., q_{t_{r_{|R|}}}) \) are the states of individual tracers. Initially there is only one start level \( R \) (since all registers are equivalent), so we define \( \overrightarrow{t_r} = \{R \rightarrow t_{r_1}\} \). Suppose the automaton is in state \( (\overrightarrow{t_r}, \overrightarrow{q}) \) and reads a constraint \( C \). Let \( L \) be the start levels of \( C \) and \( L' \) be its end levels. We define the successor state \( (\overrightarrow{t'_r}, \overrightarrow{q'}) \) and operations on the counters using the following procedure.

- To every tracer \( t_r \) that does not currently track a level, i.e. \( t_r \notin T_r \setminus \text{getTr}(L) \), the master commands \text{idle} (which causes the tracer to increment \( \text{idle}_r \)).
- For every start level \( l \in L \) that morphs into \( l' \in L' \): let \( t_r = \text{getTr}(l) \), then
  - the master sends \text{move} to \( t_r \), which causes the tracer \( t_r \) to update its counters \( C_{n_r} \) and move into a successor state \( q'_{t_r} \) (to handle \text{move}, the tracer needs a register serving as an upper bound, so the master also passes an arbitrary register from \( l \));
  - we set \( \text{getTr}'(l') = \text{getTr}(l) \), thus the tracer continues to track it.
- For every start level \( l \in L \) that disappears: let \( t_r = \text{getTr}(l) \), then
  - the master sends \text{reset} to \( t_r \), which causes the reset of the counters in \( C_{n_r} \) and the increment of \( \text{idle}_r \).
- For every new end level \( l' \in L' \):
  - we take an arbitrary \( t_r \) that is not yet mapped by \( \text{getTr}' \) and map \( \text{getTr}'(l') = t_r \);
  - the master sends \text{start} to \( t_r \).

The construction will ensure that every stable chain is tracked by a single tracer and its counter \( \text{idle} \) is bounded; and vice versa, if a tracer has its counter \( \text{idle} \) bounded, it tracks a stable chain. The acceptance of \( A \) is the formula \( \bigwedge_{t_r \in T_r} (\text{idle}_r \text{ is bounded } \rightarrow \bigwedge_{c \in C_{n_r}} (c \text{ is bounded}) \).

Tracers. We now describe the tracer component. Its goal is to track the depths of trespassing chains. When the counters of a tracer are bounded, the depths of the chains it tracks are also bounded. The tracer consists of two components, \( B_\uparrow \) and \( B_\downarrow \), which track decreasing and increasing chains. We only describe \( B_\uparrow \).

The component \( B_\uparrow \) has a set \( C_n \cup \{\text{idle}\} \) of \( |R| + 1 \) counters. A state of \( B_\uparrow \) is either the initial state \( q_0 \) or a partial mapping \( \text{getCn} : R \rightarrow C_n \). Intuitively, in each state, for
The number of counters in $C$.

**Proof.**

### A.6 Proof of Lemma 6

#### Direction $\Leftarrow$

Fix a lasso-shaped constraint sequence $C_0 \ldots C_{k-1}(C_k \ldots C_{k+1})^\omega$ having trespassing chains of unbounded depth. Since these chains have unbounded depth, they pass through $C_k$ more and more often. At moments when
the current constraint is $C_k$, each such chain is in one of the finitely-many registers. Hence there is a chain, say increasing, that on two separate occasions of reading the constraint $C_k$ goes through the same register $r$, and the chain suffix from the first pass through $r$ until the second pass has at least one $<$. Then we create an increasing chain of infinite depth by repeating this suffix forever.

**A.7 Proof of Theorem 8**

**Proof.** We first construct a nondeterministic parity automaton $A$ accepting the complement of the sought language, namely, it accepts a constraint sequence if it satisfies one of following:

- $(i)$ it is not consistent,
- $(ii)$ it has a decreasing one-way chain of infinite depth,
- $(iii)$ it has a trespassing increasing one-way chain of infinite depth, or
- $(iv)$ there is a decreasing one-way chain of depth $\geq 1$ from position 0 or $C_0|_R$ is not equal to $\{r=s \mid r,s \in R\}$.

For condition $(i)$, the automaton, by using nondeterminism, guesses a position $i$ such that $C_i|_R \neq C_{i+1}|_R$, which can be checked, again, by guessing some atom which is in $C_i|_R$ but not in $C_{i+1}|_R$, or conversely some atom not in $C_i|_R$ but in $C_{i+1}|_R$. This requires only a polynomial number $O(|R|^2)$ of states and a constant number of priorities. We now explain how to check condition $(iii)$, as conditions $(ii)$ and $(iv)$ can be checked using similar ideas.

For condition $(iii)$, the automaton needs to guess a position $i$ and a first register of the stable chain, below which there will be an infinite increasing chain, also starting from the position $i$. Starting from the position $i$, the automaton guesses a next register $t$ of the stable chain and checks that $(s=t') \in C$ belongs to the currently read constraint $C$, where $s$ is a current register representing the stable chain. We now explain how the automaton ensures the existence of a sought increasing chain. It successively guesses a sequence of registers $r_i, r_{i+1}, \ldots$ and checks that $C_j$ contains either $r_j < r'_{j+1}$ or $r_j = r'_{j+1}$ and contains $r_j < s$ for all $j \geq i$, and checks that infinitely often we see the strict $<$. This needs only $\text{poly}(|R|)$ many states and a constant number of priorities. Thus, the nondeterministic automaton $B$ has in total $\text{poly}(|R|)$ many states and $O(1)$ many priorities.

We now determinise $A$ (see e.g. [33]), which results in a deterministic automaton with $\exp(|R|)$ states and $\text{poly}(|R|)$ priorities, and complement it (no blow up). This gives us the sought automaton.

**A.8 Definition of the assumption †**

In the main text of the paper, we have defined assumption † as follows:

Every constraint $C_i$ of a given sequence $C_0C_1\ldots$ satisfies the following:

for every $\nu \in D^R, \nu' \in D^{R'}$ s.t. $\nu \cup \nu' \models C_i$, we have $|\{r' \in R' \mid \forall s \in R. \nu'(r') \neq \nu(s)\}| \leq 1$.

We now give an alternative definition. Recall that a constraint describes a set of totally ordered equivalence classes of registers from $R \cup R'$. The figure on the right describes a constraint that can be defined by the ordered equivalence classes $\{r_4, r'_4\} < \{r'_2\} < \{r_3, r'_3\} < \{r_1, r_2, r'_1\}$. It shows two columns of dots, at moment $m$ and $m+1$, where a dot describes a set of registers equivalent in that moment. The vertical levels of the dots respect the constraint: if a dot at moment $m$ is on a higher/lower/equal level than a dot at moment $m+1$, then the constraint requires the registers of the second dot to have higher/lower/equal values than the registers of the first dot. A dot on a new level appears in the second column, i.e. there were no dots on that level at moment $m$, if and only if the constraint contains an equivalence class consisting solely of $R'$-registers. A level that was present in the first column disappears from the second column if and only if the constraint contains an equivalence class consisting solely of $R$-registers. Then the assumption † is:
A.9 Proof of Lemma 9

$xy^{(m)}$-connecting chains and the exponential nature of register valuations

Fix an arbitrary 0-satisfiable constraint sequence $C_0 C_1 \ldots$ whose r2w chains are depth-bounded by $B$. Consider a moment $m$ and two registers $x$ and $y$ such that $(x > y) \in C_m$.

We would like to construct witnessing valuations $\nu_0 \nu_1 \ldots$ using the current history only, e.g. a register valuation $\nu_m$ at moment $m$ given only the prefix $C_0 \ldots C_{m-1}$. Note that the prefix $C_0 \ldots C_{m-1}$ also defines the ordered partition of registers at moment $m$, since $C_{m-1}$ is defined over $R \cup R'$. Let us see how much space we might need between $r_m(x)$ and $r_m(y)$, relying on the fact that the depths of r2w chains are bounded by $B$. Consider r2w chains that start at moment $i \leq m$ and end in $(x, m)$ (shown in blue), so it is defined within time moments $\{i, \ldots, m\}$, and l2w chains starting in $(y, m)$ and ending at moment $j \in \{i, \ldots, m\}$ (shown in pink), defined within time moments $\{j, \ldots, m\}$. Among such chains, pick one r2w and one l2w chains of depths $\alpha$ and $\beta$ that maximise the sum $\alpha + \beta$. After seeing $C_0 C_1 \ldots C_{m-1}$, we do not know how the constraint sequence will evolve in future, but by boundedness of r2w chains, any r2w chain starting in $(x, m)$ and ending in $(y, m)$ (defined within time moments $\geq m$) will have a depth $d \leq B - \alpha - \beta$ (otherwise, we could add prefix $\alpha$ and postfix $\beta$ to it and construct an r2w chain of depth larger than $B$). We conclude that $\nu_m(x) - \nu_m(y) \geq B - \alpha - \beta$, since the number of values in between two registers should be greater or equal than the longest 2w chain connecting them. To simplify the upcoming arguments, we introduce $xy^{(m)}$-connecting chains which consist of $\alpha$ and $\beta$ parts and directly connect $x$ to $y$.

An $xy^{(m)}$-connecting chain is any r2w chain of the form $(a, i) \ldots (x, m) \ldots (b, j)$: it starts in $(a, i)$ and ends in $(b, j)$, where $i \leq j \leq m$ and $a, b \in R$, and it directly connects $x$ to $y$ at moment $m$. Note that it is located solely within moments $\{i, \ldots, m\}$. Continuing the previous example, the $xy^{(m)}$-connecting chain starts with $\alpha$, directly connects $(x, m) > (y, m)$, and ends with $\beta$; its depth is $\alpha + \beta + 1$ (we have “+1” no matter how many registers are between $x$ and $y$, since $x$ and $y$ are connected directly).

With this new notion, the requirement $\nu_m(x) - \nu_m(y) \geq B - \alpha - \beta$ becomes $\nu_m(x) - \nu_m(y) \geq B - d_{xy} + 1$, where $d_{xy}$ is the largest depth of $xy^{(m)}$-connecting chains.

However, since we do not know how the constraint sequence evolves after $C_0 \ldots C_{m-1}$, we might need even more space between the registers at moment $m$. Consider an example on the right, with $R = \{r_0, r_1, r_2\}$ and the bound $B = 3$ on the depth of r2w chains.

Suppose at moment 1, after seeing the constraint $C_0$, which is $\{r_1', r_2'\} > \{r_0, r_1, r_2, r_0'\}$, the valuation is $\nu_1 = \{r_0 \mapsto 0; r_1, r_2 \mapsto 3\}$. It satisfies $\nu_1(r_2) - \nu_1(r_0) \geq B - d_{x_2} + 1$ (indeed, $B = 3$ and $d_{x_2} = 1$ at this moment); similarly for $\nu(r_1) - \nu(r_0)$.

Let the constraint $C_1$ be $\{r_1, r_2, r_3'\} > \{r_1'\} > \{r_0, r_0'\}$. What value $\nu_2(r_1)$ should register $r_1$ have at moment 2? Note that the assignment should work no matter what $C_2$ will be in future. Since the constraint $C_1$ places $r_1$ between $r_0$ and $r_2$ at moment 2, we can only assign $\nu_2(r_1) = 2$ or $\nu_2(r_1) = 1$. If we choose 2, then the constraint $C_2$ having $\{r_2, r_3'\} > \{r_1'\} > \{r_0, r_0'\}$ (the red dot in the figure) shows that there is not enough space between $r_2$ and $r_1$ at moment 2 ($\nu_2(r_2) = 3$ and $\nu_2(r_1) = 2$). Similarly for $\nu_2(r_1) = 1$: the constraint $C_2$ having $\{r_2, r_3'\} > \{r_1\} > \{r_0, r_0'\}$ (the blue dot in the figure) kills any possibilities for a correct assignment.

Thus, at moment 2, the register $r_1$ should be equally distanced from $r_0$ and $r_2$, i.e. $\nu_2(r_2) \approx \frac{\nu_2(r_0) + \nu_2(r_2)}{2}$, since its evolution can go either way, towards $r_2$ or towards $r_0$. This hints at
the exponential nature $2^{(\cdot)}$ of distances between the registers. This if formalised in the next lemma showing that any data-assignment function that places two registers $x$ and $y$ at any moment $m$ closer than $2^{m-d_{xy}}$ is doomed.

\textbf{Lemma 19 (tightness).} Fix $n \geq 3$, registers $R$ of $|R| \geq 3$, a meaningful constraint sequence prefix $C_0 \ldots C_{m-1}$ where $m \geq 1$ and whose r2w chains are depth-bounded by $b$, two registers $x, y \in R$ s.t. $(x' > y') \in C_{m-1}$, and a data-assignment function $f : (C_0 \cup C^+) \rightarrow \mathbb{N}^R$. Let $\nu_m = f(C_0 \ldots C_{m-1})$ and $d_{xy}$ be the maximal depth of $xy^{(m)}$-connecting chains. If $\nu_m(x) - \nu_m(y) < 2^{n-d_{xy}}$, then there exists a continuation $C_mC_{m+1} \ldots$ such that the whole sequence $C_0C_1 \ldots$ is meaningful and its r2w chains are depth-bounded by $b$ (hence 0-satisfiable), yet $f$ cannot satisfy it.

\textbf{Proof.} We use the idea from the previous example. The constraints $C_mC_{m+1} \ldots$ are:

1. If at moment $m$ there are registers different from $x$ and $y$, we add the step that makes them equal to $x$ (or to $y$): this does not affect the depth of $xy$-connecting chains at moments $m$ and $m+1$; also, the maximal depths of r2w chains defined at moments $\{0, \ldots, m\}$ and $\{0, \ldots, m+1\}$ stay the same. Therefore, below we assume that at moment $m$ every register is equal to $x$ or to $y$.

2. If $b - d_{xy} = 0$, we are done: $\nu_m(x) - \nu_m(y) < 2^{n-d_{xy}}$ gives $\nu_m(x) \leq \nu_m(y)$ but $C_{m-1}$ requires $\nu_m(x) > \nu_m(y)$. The future constraints then simply keep the registers constant. Otherwise, when $b - d_{xy} > 0$, we proceed as follows.

3. To ensure consistency of constraints, $C_m$ contains all atoms over $R$ that are implied by atoms over $R'$ of $C_{m-1}$.

4. $C_m$ contains $x = x'$ and $y = y'$.

5. $C_m$ places a register $z$ between $x$ and $y$: $x' > z' > y'$. This gives $d'_{xz} = d_{xy} = d_{xy} + 1 \leq b$, where $d_{xy}$ is the largest depth of connecting chains for $xy^{(m)}$, $d'_{xz} = d_{xz}^{(m+1)}$, and $d'_{zy} = d_{zy}^{(m+1)}$. Since $\nu_{m+1}(x) - \nu_{m+1}(y) < 2^{n-d_{xy}}$, either $\nu_{m+1}(x) - \nu_{m+1}(z) < 2^{n-d_{xz}}$ or $\nu_{m+1}(z) - \nu_{m+1}(y) < 2^{n-d_{zy}}$; this is the key observation. If the first case holds, we have the original setting $\nu_{m+1}(z) - \nu_{m+1}(y) < 2^{n-d_{zy}}$ but at moment $m+1$ and with registers $x$ and $z$; for the second case — with registers $x$ and $y$. Hence we repeat the whole procedure, again and again, until reaching the depth $b$, which gives the sought conclusion in item (2).

Finally, it is easy to prove that the whole constraint sequence $C_0C_1 \ldots$ is 0-satisfiable, e.g. by showing that it satisfies the conditions of Lemma 4. Moreover, it is meaningful, and all r2w chains of $C_0C_1 \ldots$ are depth-bounded by $b$ because: (a) in the initial moment $m$, all r2w chains are depth-bounded by $b$; and (b) the procedure deepens only $xy$-connecting chains and only until the depth $b$, whereas other r2w chains existing at moments $\{0, \ldots, m\}$ keep their depths unchanged (or at moments $\{0, \ldots, m+1\}$, if we executed item 1).

\textbf{Proof of Lemma 9 under additional assumption}

Tightness by Lemma 19 tells us that if a data-assignment function exists, it should separate the register values by at least $2^{n-d_{xy}}$. Such separation is sufficient as will be shown below.

We first describe a data-assignment function, then prove an invariant about it, and finally conclude with the proof of Lemma 9. For simplicity, we assume that the constraints contain a special register always holding the value 0; later we lift this assumption.

\textbf{Data-assignment function.} The function $f : (C_0 \cup C^+) \rightarrow \mathbb{N}^R$ is constructed inductively on the length of $C_0 \ldots C_{m-1}$ as follows.

Initially, $f(C_0) = \nu_0$ where $\nu_0(r) = 0$ for all $r \in R$ (recall that $C_0$ has $r = s$, $\forall r, s \in R$).
Suppose at moment \( m \), the register valuation is \( \nu_m = f(C_0)\mu C_0 \ldots C_{m-1} \). Let \( C_m \) be the next constraint, then the register valuation \( \nu_{m+1} = f(C_0)\mu C_0 \ldots C_m \) is:

1. Let \( \nu_m(x) = [\nu_m(a) + \nu_m(b)]/2 \). By data-assignment function (item 3), \( \nu_m(a) = \nu_{m+1}(x) \) and \( \nu_m(b) = \nu_{m+1}(y) \). Note that the number of levels between \( x \) and \( y \) may differ. This is shown on the right picture. Consider the depths of connecting chains for \( ab^m \) and \( xy^{m+1} \): Since every \( ab^m \)-connecting chain can be extended to \( xy^{m+1} \)-connecting chain of the same depth as shown on the figure, we have \( d_{ab} \leq d_{xy}^{1} \), and hence \( 2^{n-d_{ab}} \geq 2^{n-d_{xy}} \). Using the inductive hypothesis, we conclude \( \nu_{m+1}(x) - \nu_{m+1}(y) = \nu_m(a) - \nu_m(b) \geq 2^{n-d_{ab}} \geq 2^{n-d_{xy}} \).

### Case 1: both present.

The levels of \( x \) and \( y \) at \( m+1 \) also exist at moment \( m \). Let \( a, b \) be registers s.t. \( a > b \) in \( C_m \) at moment \( m \) on the same levels as \( x \) and \( y \) at moment \( m+1 \). By data-assignment function (item 3), \( \nu_m(a) = \nu_{m+1}(x) \) and \( \nu_m(b) = \nu_{m+1}(y) \). Note that the number of levels between \( x \) and \( y \) may differ. This is shown on the right picture. Consider the depths of connecting chains for \( ab^m \) and \( xy^{m+1} \): Since every \( ab^m \)-connecting chain can be extended to \( xy^{m+1} \)-connecting chain of the same depth as shown on the figure, we have \( d_{ab} \leq d_{xy}^{1} \), and hence \( 2^{n-d_{ab}} \geq 2^{n-d_{xy}} \). Using the inductive hypothesis, we conclude \( \nu_{m+1}(x) - \nu_{m+1}(y) = \nu_m(a) - \nu_m(b) \geq 2^{n-d_{ab}} \geq 2^{n-d_{xy}} \).

### Case 2: \( x \) is new top.

The register \( x \) at moment \( m+1 \) is on a new level that is between the levels of \( a \) and \( b \) at moment \( m \), so \( \nu_{m+1}(x) = [\nu_m(a) + \nu_m(b)]/2 \). The register \( y \) at moment \( m+1 \) is on a level that was also present at moment \( m \), witnessed by register \( c \). Formally, \( C_m \) contains \( a > x' > b \) at moment \( m \), \( c = y' \), and \( x' > x' \). Note that \( c \) and \( b \) may coincide.

Then, \( \nu_{m+1}(x) - \nu_{m+1}(y) = [\nu_m(a) + \nu_m(b)]/2 - \nu_m(c) = [\nu_m(a) - \nu_m(c)] + [\nu_m(b) - \nu_m(c)] \geq \nu_m(a) - \nu_m(c) + [\nu_m(b) - \nu_m(c)] \geq [\nu_m(a) - \nu_m(c)] + [\nu_m(b) - \nu_m(c)] \geq [2^{n-d_{ac}}] + [2^{n-d_{bc}}] \geq 2^{n-d_{ac}} + 2^{n-d_{bc}} \); the latter holds because \( d_{ac} < b \) while \( d_{bc} \leq b \). We need to prove that the last sum is greater or equal to \( 2^{n-d_{xy}} \). The picture shows how the green \( xy^{m+1} \)-connecting chain

\[ \frac{d_{ab}}{d_{xy}} \]

A stronger result holds, namely \( d_{ab} = d_{xy} \), but it is not needed here.
can be constructed from the pink ac\((m)\)-connecting chain, hence \(d_{xy} \geq d_{ac} + 1\), so we get 
\[2^{n-d_{ac}-1} \geq 2^{n-d_{cy}}.\] 
Hence, \(\nu_{m+1}(x) - \nu_{m+1}(y) \geq 2^{n-d_{ac}-1} + \lfloor 2^{n-d_{ac}-1}\rfloor \geq 2^{n-d_{cy}}.\)

**Case 4:** \(x\) was present, \(y\) is middle new. The case is similar to the previous one, but we prove it for completeness. The constraint \(C_m\) contains \(a = a', x' > y', b > y' > c\), where \(b\) and \(c\) are adjacent (\(a\) and \(b\) might be the same). Then, 
\[\nu_{m+1}(x) - \nu_{m+1}(y) = \nu_m(a) - \left(\nu_m(b) + \nu_m(c)\right) \geq \frac{\nu_m(a) - \nu_m(b)}{2} + \frac{\nu_m(s) - \nu_m(c)}{2} \geq \left[2^{n-d_{ac}-1}\right] + \frac{2^{n-d_{ac}-1}}{2} \geq \left[2^{n-d_{ac}-1}\right] + 2\left[2^{n-d_{ac}-1}\right] \geq 2^{n-d_{ac}-1} + 2^{n-d_{ac}-1} = 2^{n-d_{cy}},\]
and since \(d_{ac} + 1 \leq d_{xy}\), we get \(\nu_{m+1}(x) - \nu_{m+1}(y) \geq 2^{n-d_{cy}}.\)

**Proof of Lemma 9.** It is sufficient to show that for every atom \((r \ni s)\) or \((r \ni s')\) of \(C_m\), where \(r, s \in R\) and \(\ni \in \{<, >, =\}\), the expressions \(\nu_m(r) \ni \nu_m(s)\) or \(\nu_m(r) \ni \nu_m(s)\) hold, respectively. Depending on \(r \ni s\), there are the following cases.

- If \(C_m\) contains \((r = s)\) or \((r = s')\) for \(r, s \in R\), then item (3) implies resp. \(\nu_m(r) = \nu_m(s)\) or \(\nu_m(r) = \nu_m(s)\) by the invariant.

- If \((r > s) \in C_m\), then \(\nu_m(r) > \nu_m(s)\) by the invariant.

- Let \((r > s') \in C_m\) and the level of \(s\) at moment \(m + 1\) be present at moment \(m\), i.e. there is a register \(t\) such that \((t = s') \in C_m\). Since \(\nu_m(t) = \nu_{m+1}(s)\) by item (3) and since \(\nu_m(r) > \nu_m(t)\) by \((r > t = s') \in C_m\), we get \(\nu_m(r) > \nu_{m+1}(s)\). Similarly for the case \((r < s') \in C_m\) where \(s\) lays on a level also present at moment \(m\).

- Let \((r > s') \in C_m\) and \(s\) lays on the highest level among all levels at moments \(m\) and \(m + 1\). Then \(\nu_m(r) < \nu_{m+1}(s)\) because \(\nu_{m+1}(s) \geq \nu_m(r) + 2^n\) by item (1).

- Finally, there are two cases left: \((r > s') \in C_m\) or \((r < s') \in C_m\), where \(s\) lays on a newly created level at moment \(m + 1\), and there are higher levels at moment \(m\). This corresponds to item (2). Let \((a > b) \in C_m\) be two adjacent registers at moment \(m\) between which the register \(s\) is inserted at moment \(m + 1\), so \((a > s') > (b, m) \in C_m\). Let \(d_{ab}\) be the maximal depth of \(ab(m)\)-connecting chains; fix one such chain. We change it by going through \(s\) at moment \(m + 1\), i.e. substitute the part \((a, m) > (b, m)\) by \((a, m) > (s, m + 1) > (b, m)\): the depth of the resulting chain is \(d_{ab} + 1\) and it is \(\leq B\) by boundedness of \(r2w\) chains. Hence \(d_{ab} \leq B - 1\), so \(\nu_m(a) - \nu_m(b) \geq 2\), implying \(\nu_m(a) > \frac{\nu_m(a) + \nu_m(b)}{2} > \nu_m(b)\). When \((r > s') \in C_m\) we get \(\nu_{m+1}(r) \geq \nu_m(a)\), and when \((r < s') \in C_m\) we get \(\nu_{m+1}(r) \leq \nu_m(b)\), therefore we are done.

Finally, the function always assigns nonnegative numbers, from \(\mathbb{N}\), so we are done.

**Lifting the assumption about 0**

In this section we will lift the assumption about a register always holding 0.

**Conversion function.** Given a meaningful constraint sequence \(C_0 C_1 \ldots C_n\) over \(R\) without a special register holding 0, we will construct, on-the-fly, a meaningful sequence \(\tilde{C}_0 \tilde{C}_1 \ldots\) that has such a register; call the register \(r_0 \notin R\) and let \(\tilde{R}_0 = R \cup \{r_0\}\). Intuitively, we will add atoms \(r = r_0\) only if they follow from what is already known and otherwise add atoms \(r > r_0\).

Initially, in addition to the atoms of \(C_0\), we require \(r = r_0\) for every \(r \in R\) (recall that the original \(C_0\) contains \(r_1 = r_2\) for all \(r_1, r_2 \in R\)). This gives an incomplete constraint \(\tilde{C}_0\) over \(\tilde{R}_0 \cup \tilde{R}_0'\): it does not yet have atoms of the form \(r \ni r'_0\), \(r_0 \ni r', r'_0 \ni r'\), where \(r \in \tilde{R}_0\).

At moment \(m \geq 0\), given a constraint \(\tilde{C}_m|R_0\) over \(\tilde{R}_0\) (without primed registers \(R'_0\)) and a constraint \(C_m\) over \(R \cup R'\) (without register \(r_0\)), we construct \(\tilde{C}_m\) over \(R_0 \cup R'_0\) as follows:

- \(\tilde{C}_m\) contains all atoms of \(C_m\).
- \((r_0 = r'_0) \in \tilde{C}_m\).

\[\tilde{C}_m = C_m \cup \{r = r_0\}\]
For every \( r \in R \): if \( r' = r_0 \) is implied by the current atoms of \( \tilde{C}_m \), then we add it, otherwise we add \( r' > r_0 \).

Notice that the atom \( r' < r_0 \) is never implied by \( \tilde{C}_m \), as we show now. Suppose the contrary. Then, since \( C_m \) does not talk about \( r_0 \) or \( r'_0 \), there should be \( s \in R \) such that \( (s = r_0) \in \tilde{C}_m | r_0 \) and \( (r' < s) \in C_m \). Because \( (s = r_0) \in \tilde{C}_m | r_0 \) happens iff there is a 1w chain \( (r_1, 0) = (r_2, 1) = \ldots = (s, m) \) of zero depth\(^2\), we can construct the 1w decreasing chain \( (r_1, 0) = (r_2, 1) = \ldots = (s, m) > (r, m + 1) \) of depth 1, which implies that \( C_0 C_1 \ldots \) is not 0-consistent. Hence our assumption is wrong and \( (r' < r_0) \in \tilde{C}_m \) is not possible.

Finally, to make \( \tilde{C}_m \) maximal, we add all atoms implied by \( \tilde{C}_m \) but not present there. Using this construction, we can easily define \( C0nv : C^+ \rightarrow \tilde{C} \) and map a given meaningful constraint sequence \( C_0 C_1 \ldots \) to \( \tilde{C}_0 \tilde{C}_1 \ldots \) with a dedicated register holding 0. Notice that the constructed sequence is also meaningful, because we never add inconsistent atoms and never add an atom \( r' < r_0 \) (see the third item). Finally, in the constructed sequence the depths of r2w chains can increase by at most 1, due to the register \( r_0 \): it can deepen-by-one a finite chain, unless the chain is already ending in a register holding 0. Hence we got the following lemma.

**Lemma 20.** For every meaningful constraint sequence \( C_0 C_1 \ldots \), the sequence \( \tilde{C}_0 \tilde{C}_1 \ldots \) constructed with \( C0nv \) is also meaningful. Moreover, the maximal depth of r2w chains cannot increase by more than 1.

Finally, we lift the assumption about a special register. Using \( C0nv \), we translate a given meaningful constraint sequence prefix \( C_0 \ldots C_m \) into \( \tilde{C}_0 \ldots \tilde{C}_m \) that contains a register always holding 0. Now we can apply the data-assignment function as described before. By definition of \( C0nv \), the original constraint \( C_i \subset \tilde{C}_i \) for every \( i \geq 0 \), so the resulting valuation satisfies the original constraints as well. This concludes the proof of Lemma 9.

### B Proofs of Section 3

#### B.1 Proof of Theorem 10

**Proof idea.** We reduce the problem from the halting problem of 2-counter machines, which is undecidable [32]. We define a specification with 4 registers \( r_1, r_2, z \) and \( t \). \( r_1 \) and \( r_2 \) each store the value of one counter; \( z \) stores 0 to conduct zero tests and \( t \) is used as a buffer. We now describe how to increment \( c_1 \) (cf Figure 6a): the case of \( c_2 \) and of decrementing are similar. Eve suggests a value \( d > r_1 \), which is stored in \( t \). Then, Adam can check that the increment was done correctly: Eve cheated if and only if he can provide a data \( d' \) such that \( r_1 < d' < d \). If he cannot, \( d \) is stored in \( r_1 \), thus updating the value of the counter. The acceptance condition is then a reachability one, asking that a halting instruction is eventually met. Now, if \( M \) halts, then its run is easily simulated by a strategy of Eve. Conversely, if \( M \) does not halt, then no halting instruction is reachable by simulating \( M \) correctly, and Adam is able to check that Eve does not cheat during its simulation.

**Remark 21.** As a matter of fact, if \( M \) halts, then its run is finite and the values of the counters are bounded by some \( B \), so Eve’s strategy can even be modelled using a transducer with \( B \) registers, which simulates the run by providing the values of the counters along the run. This shows that the transducer synthesis problem from specifications defined by register

\(^2\) The proof of this claim is omitted.
automata (without the input-driven restriction) is undecidable (cf Appendix B.6 for the formal definitions of those objects).

**Proof.** We reduce from the halting problem of deterministic 2-counter machines, which is undecidable [32]. Among multiple formalisations of counter machines, we pick the following one: a 2-counter machine has two counters which contain integers, initially valued 0. It is composed of a finite set of instructions $M = (I_1, \ldots, I_m)$, each instruction being of the form $\text{inc}_j, \text{dec}_j, \text{ifz}_j(k', k'')$ for $j = 1, 2$ and $k', k'' \in \{1, \ldots, m\}$, or halt. The semantics are defined as follows: a configuration of $M$ is a triple $(k, c_1, c_2)$, where $1 \leq k \leq m$ and $c_1, c_2 \in \mathbb{N}$. The transition relation (which is actually a function, as $M$ is deterministic) is then, from a configuration $(k, c_1, c_2)$:

- If $I_k = \text{inc}_1$, then the machine increments $c_1$ and jumps to the next instruction $I_{k+1}$: $(k, c_1, c_2) \rightarrow (k+1, c_1, c_2)$. Similarly for inc$_2$.
- If $I_k = \text{dec}_1$ and $c_1 > 0$, then $(k, c_1, c_2) \rightarrow (k+1, c_1-1, c_2)$. If $c_1 = 0$, then the computation fails and there is no successor configuration. Similarly for dec$_2$.
- If $I_k = \text{ifz}_1(k', k'')$, then $M$ jumps to $k'$ or $k''$ according to a zero-test on $c_1$: if $c_1 = 0$, then $(k, c_1, c_2) \rightarrow (k', c_1, c_2)$; otherwise $(k, c_1, c_2) \rightarrow (k'', c_1, c_2)$. Similarly for ifz$_2$.

A run of the machine is then a finite or infinite sequence of successive configurations, starting at $(1, 0, 0)$. We say that $M$ **halts** whenever it admits a finite run which ends in a configuration $(k, c_1, c_2)$ such that $I_k = \text{halt}$.

Let $M = (I_1, \ldots, I_m)$ be a 2-counter machine. We associate to it the following DRA specification: $S$ has states $Q = \{0, \ldots, m+1\} \times \{i, o, y, n\} \cup \{\dagger, \ddagger\}$, and has four registers $r_1, r_2, t, z$. $i$ and $o$ respectively denote input and output states, while $y$ and $n$ are used to remember whether an ifz test evaluated to true or false. The initial state is $(0, i)$, and acceptance is defined by a reachability condition with $F = \{\dagger\}$, and $\dagger$ is a sink rejecting state. Its transitions are as follows:

- Initially, there is a transition $(0, i) \xrightarrow{T} (1, o)$ so that the implementation can start the simulation.
- Then, for each $k \in \{1, \ldots, m\}$:
  - If $I_k = \text{inc}_j$ for $j = 1, 2$, then we add the gadget of Figure 6a, i.e. output transition $(k, o) \xrightarrow{r_1, t} (k, i)$ and input transitions $(k, i) \xrightarrow{r_1} \dagger$, $(k, i) \xrightarrow{t} (k+1, o)$ and $(k, i) \xrightarrow{c_1} \dagger$, $(k, i) \xrightarrow{c_2} \dagger$.
  - The case $I_k = \text{dec}_j$ for $j = 1, 2$ is similar: we add output transition $(k, o) \xrightarrow{r_1, t} (k, i)$ and input transitions $(k, i) \xrightarrow{r_1} \dagger$, $(k, i) \xrightarrow{t} (k+1, o)$ and $(k, i) \xrightarrow{c_1} \dagger$, $(k, i) \xrightarrow{c_2} \dagger$. Note that in our definition, if $c_j = 0$, then the instruction dec$_j$ should be

**Figure 6** Simulating increment (the gadget for decrementing is similar) and ifzero tests.
As a consequence, which ignores its input and outputs configuration reached by $u$ (1 of the counter modified or tested at step).

Now, assume that $M$ admits an accepting run $\rho = (k_1, c_1, c_1^1) \rightarrow \cdots \rightarrow (k_n, c_n^1, c_n^2)$, where $n \in \mathbb{N}$, $k_1 = 1$, $c_1^1 = c_1^2 = 0$ and $I_{k_i} = \text{halt}$. We can then define a strategy $\lambda_E$ of Eve in $G$, which ignores its input and outputs $w = c_0^0 \cdots c_{n-1}^0 0^\omega$, where for $1 \leq l < n$, $j_l$ is the index of the counter modified or tested at step $l$ (i.e. $j_l = 1, 2$ is such that $I_{k_i} = \text{inc}_{j_l}, \text{dec}_{j_l}$ of $\text{ifz}_{j_l}(k', k''_l)$).

Let us show that $\lambda_E$ is indeed a winning strategy: let $u \in \mathbb{N}^\omega$ be a word input by Adam. We show by induction on $l$ that in $S$ the partial run over $(u \otimes w)[l]$ is either in state $\tilde{f}$ or $S$ is in configuration $(k_1, \tau_1)$, where $\tau_1(\tau_1) = c_1^2$ and $\tau_1(\tau_2) = c_1^2$. Initially, $S$ is in configuration $(1, \tau_1^0)$, so the invariant holds. Now, assume it holds up to step $l$. If $S$ is in $\tilde{f}$, the invariant holds at step $l + 1$ as $\tilde{f}$ is a sink state. Otherwise, necessarily $l < n$, $S$ is in configuration $(k_1, \tau_1)$ and there are four cases:

- $I_{k_i} = \text{inc}_{j_i}$. By definition, $j = j_i$. We treat the case $j = 1$, the other case is similar. Then, Eve outputs $c_1^1 = c_1^1 \tau_1^1 + 1$, which is such that $c_1^1 > \tau_1(\tau_1)$. Then, there does not exist $d$ such that $\tau_1(\tau_1) < d < \tau_1(t)$ since $\tau_1(\tau_1) = c_1^1 - 1$ and $\tau_1(t) = c_1^1 + 1$, so the transition to $\tilde{f}$ cannot be taken. Now, either $u[l + 1] = \tau_1(t) = c_1^1 + 1$, in which case $S$ evolves to configuration $(k_1, c_1^1, c_1^2_{l+1})$, or $u[l + 1] \neq \tau_1(t)$ and $S$ goes to $\tilde{f}$; in both cases the invariant holds.

- The case of $I_{k_i} = \text{dec}_{j_i}$ is similar. Let us just mention that the computation does not block at this step, otherwise $\rho$ is not a run of $M$, so the transition $d < r_j$ can indeed be taken.

- $I_{k_i} = \text{ifz}_{j_i}(k', k''_i)$. Again, $j = j_i$, and we treat the case $j = 1$. Eve outputs $c_1^1$; there are two cases. If $c_1^1 = 0$, the transition $* = r_1 \land * = z$ is taken, since at every step, $\tau_1(z) = 0$ (this register is never modified). If $c_1^1 \neq 0$, then transition $* = r_1 \land * = z$ is taken. In both cases, whatever the input, $S$ then evolves to $(k_1, \tau_1^{1+1})$ (where $\tau_1^{1+1} = \tau_1$) and the invariant holds.

- Finally, if $I_{k_i} = \text{halt}$, then whatever the output, $S$ transitions to $\tilde{f}$.

As a consequence, $\tilde{f}$ is eventually reached whatever the input, which means that for all $u \in \mathbb{N}^\omega$, $u \otimes I(u) \in S$, i.e. Eve indeed wins $G$.

Conversely, assume that Eve has a winning strategy $\lambda_E$ in $G$. Let $\rho$ be the maximal run of $M$ (i.e. either $\rho$ ends in a configuration with no successor, or it is finite). It is unique since $M$ is deterministic. Let $n = \|\rho\|$, with the convention that $n = \infty$ if $\rho$ is infinite. Let us build by induction an input word $u$ such that for all $l < n$, $\lambda_E(u)[l] = c_0^l$ and the configuration reached by $S$ over $(u \otimes I(u))[l]$ is $(k_1, \tau_1)$. Initially, let $u[0] = 0$. As the initial test is $\top$, $S$ anyway evolves to state $(1, 0)$, with $\tau_1(\tau_1) = \tau_2(\tau_2) = 0$.

Now, assume we built such input $u$ up to $l$. There are again four cases:

- $I_{k_i} = \text{inc}_{j_i}$. Then Eve provides some output data $d_0 > \tau_1(\tau_1)$. Assume $d_0 > \tau_1(\tau_1) + 1$. Then, Eve loses because on reading input data $d_1 = \tau_1(\tau_1) + 1$, $S$ goes to state $\tilde{f}$, which is a sink rejecting state. So, necessarily, $d_0 = \tau_1(\tau_1) + 1 = c_1^0$, and $S$ evolves to configuration $(k_1, \tau_1 + 1, \tau_1 + 1)$.

- The case $I_{k_i} = \text{dec}_{j_i}$ is similar. Necessarily, $c_1^1 > 0$, otherwise Eve cannot provide any output data and is thus losing. Thus, the computation does not block here.

- $I_{k_i} = \text{ifz}_{j_i}(k', k''_i)$. The output transitions of the gadget constrains Eve to output $d_1 = \tau_1(\tau_1) = c_1^0$, and $S$ evolves to configuration $(k_1, \tau_1 + 1, \tau_1 + 1)$.
B.2 Proof of Lemma 13

Given \( \pi, \text{tst}, \text{asgn} \), we define the mapping \( \text{constr} : (\pi, \text{tst}, \text{asgn}) \rightarrow C \) as follows. (The definition is as expected, but we should be careful about handling of \( r_d \), it is the last item.)

- The constraint \( C \) includes all atoms of the state constraint \( \pi \) (that relates the registers at the beginning of the step).
- Recall that neither \( \text{tst} \) nor \( \text{asgn} \) talk about \( r_d \). For readability, we shorten \( (t_1 \otimes t_2) \in C \) to simply \( t_1 \otimes t_2 \), \((s \otimes r) \in \text{tst} \) to \( s \otimes r \), and \( a \leq b \) means \((a < b) \vee (a = b)\).
- We define the order at the end of the step as follows. For every two different \( r, s \in R \):
  - \( r' = s' \) iff \( (r = s) \land r, s \notin \text{asgn} \lor (s = s) \lor r, s \in \text{asgn}; \)
  - \( r' < s' \) iff \( (r < s) \land r, s \notin \text{asgn} \lor (s < s) \land r \in \text{asgn} \lor s \notin \text{asgn} \);
  - \( r' = r' \) iff \( (r = s) \lor r \in \text{asgn} \);
  - \( r' \otimes r' \) iff \( (r \otimes s) \lor r \notin \text{asgn} \), for \( \otimes \in \{<,>\} \).
- So far we have defined the order of the registers at the beginning and the end of the step.
- Now we relate the values between these two moments. For every \( r \in R \):
  - \( r = r' \) iff \( r \notin \text{asgn} \lor r \in \text{asgn} \land (s = r) \);
  - \( r \otimes r' \) iff \( r \in \text{asgn} \land (r \otimes s) \), for \( \otimes \in \{<,>\} \);
- Finally, we relate the values of \( r_d \) between the moments. There are two cases.
  - The value of \( r_d \) crosses another register: \( \exists r \in R: (r_d < r) \land (s \geq r) \). Then \( (r'_d > r_d) \).
  - Similarly for the opposite direction: if \( \exists r \in R: (r_d > r) \land (s \leq r) \) then \( (r'_d < r_d) \).
  - Otherwise, the value of \( r_d \) does not cross any register boundary. Then \( r'_d = r_d \).

Using the mapping \( \text{constr} \), every action word \( \bar{a} = (\text{tst}_0, \text{asgn}_0)(\text{tst}_1, \text{asgn}_1) \ldots \) is uniquely mapped to the constraint sequence \( C_0 C_1 \ldots \) as follows: \( C_0 = \text{constr}(\pi_0, \text{tst}_0, \text{asgn}_0) \), set \( \pi_1 = \text{unprime}(C_0 | R_d) \), then \( C_1 = \text{constr}(\pi_1, \text{tst}_1, \text{asgn}_1) \), and so on.

We now prove that an action word is feasible iff the constructed constraint sequence is 0-satisfiable. This follows from the definitions of feasibility and 0-satisfiability, and from the following simple property of feasible action words. Every feasible action word has a witness \( \nu_0 \nu_1 \nu_2 \ldots \in (\mathbb{D}^R \cdot \mathbb{D})^\omega \) such that: if some \( \text{tst} \) is repeated twice and no assignment is done, then the value \( d \) stays the same. This property is needed due to the last item in the definition of \( \text{constr} \) where we set \( r'_d = r_d \).

B.3 Proof of Lemma 14

First, we show that the set \( \text{Feasible}_\omega (R) \) is definable by a deterministic parity or max-automata for \( \mathbb{Q} \) and \( \mathbb{N} \), respectively. The lemma then follows from the facts that parity automata and max-automata are closed under Boolean operations, and deterministic max-automata can express all \( \omega \)-regular languages [7].

We describe a deterministic (parity or max) automaton \( A' \) accepting all feasible action words. Let \( A \) the deterministic (parity or max) automaton accepting all 0-satisfiable constraint sequences (see Theorems 1 or 5). Our automaton \( A' \) in its state \((q, \pi)\) tracks the state \( q \) of \( A \) and the state constraint \( \pi \). The initial state of \( A' \) is \((q_0, \pi_0)\), where \( q_0 \) is initial for \( A \) and \( \pi_0 = \{ r = s | r, s \in R_d \} \). From a state \((q, \pi)\), on reading \((\text{tst}, \text{asgn})\), the automaton creates the constraint \( C = \text{constr}(\pi, \text{tst}, \text{asgn}) \) (by Lemma 13), simulates \( A \) on reading \( C \), which gives \( q' \), and updates \( \pi' = \text{unprime}(C | R_d) \); thus, \( A' \) transits into \((q', \pi')\). The acceptance is defined
by the acceptance of $A$. It is easy to see that the automaton $A'$ accepts an action word if it is feasible. State constraints can be represented as functions $\pi : R \times R \rightarrow \{<, >, =\}$, so overall the size of $A'$ is $2^{q_0|\lambda R|}$. As a consequence, $W_f$ is recognised by the product of $S$ and the complement of $A'$, which is an automaton of size $O(|Q|2^{q_0|\lambda R|})$.

### B.4 Proof of Theorem 11 for $(Q, \leq)$

We extend the proof idea of Theorem 11 for $(Q, \leq)$ sketched on page 11. The theorem essentially follows from the two propositions below:

#### Proposition 22. If Eve wins $G_f$, then she wins $G$.

**Proof.** Let $\lambda_E^f : (V_0 V_3)^+ \rightarrow V_2$ be a winning Eve strategy in $G_f$. We construct a winning Eve strategy $\lambda_E : Tst^+ \rightarrow \Sigma$ in $G$ as follows\(^3\). Fix an arbitrary sequence $\text{tst}_0...\text{tst}_k$; we are going to define $\lambda(\text{tst}_0...\text{tst}_k)$. First, for all $0 \leq i \leq k-1$, we inductively define $v_0, u_0, v_1, u_1, \ldots, v_k \in (Q A \cup Q E)$, $\text{asgn}_0,...,\text{asgn}_k$, and $\sigma_1,...,\sigma_k \in \Sigma$:

- The state $v_0$ is initial for the register automaton $G$.
- For all $0 \leq i \leq k$, define $u_i \in Q_E$ and $\text{asgn}_i$ to be such that $(\text{asgn}_i, u_i) = \delta(v_i, \text{tst}_i)$, $\sigma_{i+1} = \lambda_E^f(v_0(\text{tst}_0, \text{asgn}_0, u_0)(\sigma_1, v_1))...\text{tst}_i, \text{asgn}_i, u_i))$, and $v_{i+1} = \delta(u_i, \sigma_i)$.

We then set $\lambda_E(\text{tst}_0...\text{tst}_k) = \sigma_{k+1}$. We now show that the constructed Eve strategy $\lambda_E$ is winning in $G$. Consider a Adam data strategy $\lambda_A^f$, and let $(v_0, v_0)(u_0, v_1)(v_1, v_2)...$ be an infinite run in $G$ on reading the outcome $\lambda_E^f || \lambda_E$; it is enough to show that $v_0 u_0 v_1 u_1...$ satisfies the parity condition. Let $d_0 d_1...$ be the sequence of data produced by Adam during the play, let $\sigma_0 \sigma_1...$ be the labels produced by Eve strategy $\lambda_E$, and let $\overline{\pi} = (\text{tst}_0, \text{asgn}_0)(\text{tst}_1, \text{asgn}_1)...$ be the tests and assignments performed by the automaton during the run. It is easy to see that the sequence $v_0(\text{tst}_0, \text{asgn}_0, u_0)(\sigma_0, v_1)(\text{tst}_1, \text{asgn}_1, u_1)...$ constitutes a play in $G_f$, and it is compatible with $\lambda_E^f$. Also, the action word $\overline{\pi}$ is feasible (which is witnessed by $v_0 d_0 d_1 d_1...$). Therefore, since $\lambda_E^f$ is winning, the sequence $v_0 u_0 v_1 u_1...$ satisfies the parity condition. 

#### Proposition 23. If Adam wins $G_f$, then Adam wins $G$.

**Proof.** Given an Adam winning strategy $\lambda_A^f : V_0 (V_0 V_3)^* \rightarrow V_3$, we construct the winning Adam data strategy $\lambda_A^f$ in $G$ step-by-step as follows. Suppose we are in the middle of a play: $d_0...d_{k-1}$ has been played by Adam $\lambda_A^f$ and $\sigma_0...\sigma_{k-1}$ has been played by Eve; both sequences are empty initially. We want to know the value $d_k$ for $\lambda_A^f(\sigma_0...\sigma_{k-1})$. Let $(v_0, v_0)(u_0, v_1)(v_1, v_2)...(v_k, v_k)$ be the current run prefix of the register automaton $G$ (initially $(v_0, v_0)$). Let $v_0(\text{tst}_0, \text{asgn}_0, u_0)(\sigma_0, v_1)(\text{tst}_1, \text{asgn}_1, u_1)...(\sigma_{k-1}, v_k)$ be the corresponding play prefix of $G_f$ (initially $v_0$). We assume that this play prefix adheres to $\lambda_A^f$ (this holds initially). We now consult $\lambda_A^f$; let $(\text{tst}_k, \text{asgn}_k, u_k) = \lambda_A^f(\sigma_{k-1}, v_k)$. Using $\text{tst}_k$ and $u_k$, we construct $d_k$ as follows.

- If $\text{tst}_k$ contains $* = r$ for some $r \in R$, we set $d_k = \nu_k(r)$.
- If $\text{tst}_k$ is of the form $r < *$ for all $r \in R$, then set $d_k = \max(\nu_k) + 1$, i.e. take the largest value held in the registers plus 1.
- Similarly, if $\text{tst}_k$ is of the form $* < r$ for all $r \in R$, then set $d_k = \min(\nu_k) - 1$.

---

\(^3\) What we really need is a winning Eve strategy of the form $\lambda_E^0 : \emptyset \rightarrow \Sigma$. The strategy $\lambda_E : Tst^+ \rightarrow \Sigma$ that we construct encodes $\lambda_E^0$ as follows: it has the same set $R$ of registers as the automaton $G$, and performs the same assignment actions as the automaton. Then, on seeing a new data, it compares the data with the register values, which induces a test, and passes this test to $\lambda_E$. 
Otherwise, for every \( r \in R \), the test \( \text{tst}_k \) has either \( r < \ast \) or \( \ast < r \). We now pick two registers \( r, s \) such that the test contains \( r < \ast \) and \( \ast < s \) and no register holds a value between \( \nu_k(r) \) and \( \nu_k(s) \). Then we set \( d_k = \frac{\nu_k(r) + \nu_k(s)}{2} \).

Then, \( d_k \) satisfies \( \text{tst}_k \), i.e. \( (\nu_k, d_k) \models \text{tst}_k \). Finally, define \( \nu_{k+1} = \text{update}(\nu_k, d_k, \text{asgn}_k) \). Thus, the next configuration of the run in the register automaton is \( (u_k, \nu_{k+1}) \). In \( G_f \), the play is extended by \( (\text{tst}_k, \text{asgn}_k, u_k) \); notice that the resulting extended play again adheres to the winning Adam strategy \( \lambda'_A \). Therefore, starting from the empty sequences of Adam data choices and Eve label choices, step-by-step we construct the values for \( \lambda'_A \). The resulting outcome in the Church game \( G \) induces a rejecting run in the register automaton because Adam wins the corresponding play in \( G_f \).

Now, we are ready to prove Theorem 11:

**Proof.** First, we show that Eve wins \( G \) iff she wins \( G_f \). Direction \( \Rightarrow \) follows from Proposition 22. Direction \( \Leftarrow \) is proven by contraposition relying on the determinacy of \( \omega \)-regular games and Proposition 23. Since the feasibility game \( G_f \) is of size polynomial in \( |Q| \) and exponential in \( |R| \), and has a number of priorities polynomial in \( c \), it can be solved in \( O(|poly(|Q|)|2^{poly(|R|)})poly(|c|) \) and we are done with the first item of the theorem. The determinacy is proven similarly using the claims above and the determinacy of \( \omega \)-regular games.

Finally, Remark 12 on finite-memoriness follows from the proof of claim (1), where we have built the strategy \( \lambda_E : \text{Tst}^+ \rightarrow \Sigma \), and from the finite-memoriness of \( \omega \)-regular games.

### B.5 Data strategy for Adam (proof of Proposition 15, direction \( \Rightarrow \))

**Lemma 24 (data strategy).** Let \( G \) be a Church game. If Adam wins \( G^{	ext{reg}}_f \), then he wins \( G \).

This seemingly easy lemma is not trivial. Despite Adam having in \( G^{	ext{reg}}_f \) a winning strategy \( \lambda'_A : V_f(V_3V_5)^* \rightarrow V_3 \), which can also be expressed in the form \( \lambda_A : \Sigma^* \rightarrow \text{Tst} \), it is not clear how to instantiate it to a data strategy \( \lambda'_A : \Sigma^* \rightarrow \Sigma \). For instance, if the strategy \( \lambda_A \) in \( G^{	ext{reg}}_f \) dictates Adam to pick the test \( \ast > r \), which data should \( \lambda'_A \) pick: \( \nu(r) + 1 \), \( \nu(r) + 2 \), more? For different Eves different values may be needed. We will show how to construct \( \lambda'_A \) from a given \( \lambda_A \) that beats *any* Eve. The steps are the following:

- In Section 2, we defined so-called r2w chains: they track how many values have been inserted between two registers so far. We show that if Adam wins the register game, there must be an upper bound \( B \) on the number of such insertions (because \( \Sigma \) is not dense).
- Furthermore, in that section, we have studied constraint sequences whose r2w chains are bounded. We described, for any given bound \( B \), a data-assignment function that on-the-fly assigns data values to the registers that satisfy the constraints. ‘On-the-fly’ means that the function reads the constraint sequence and at any given moment relies only on the history of constraints and data values, but does not see the future.
- Thus, we can construct \( \lambda'_A \) from \( \lambda'_A \) by translating the currently played action-word prefix \( \text{tst}_0, \text{asgn}_0 \ldots \text{tst}_m, \text{asgn}_m \) into a constraint-sequence prefix and applying the data-assignment function to it.

**Boundedness of right two-way chains induced by Adam**

Suppose Adam wins \( G^{	ext{reg}}_f \) using a finite-memory strategy \( \lambda'_A : V_f(V_3V_5)^* \rightarrow V_3 \) (equiv., \( \lambda'_A : \Sigma^* \rightarrow \text{Tst} \)). The plays consistent with \( \lambda'_A \) satisfy the following important property. Fix a moment \( i \) and a register \( x \). Then: after the moment \( i \), only a bounded number of values...
can be inserted below the value of register \( x \) at moment \( i \). Similarly, if we fix two registers at moment \( i \), there can only be a bounded number of insertions between the values of \( x \) and \( y \) at moment \( i \). This holds because, by finiteness of Adam strategy, once the number of such insertions is larger than the memory of Adam, Eve can repeat her actions to force an infinite number of such insertions, leading to a play with an unfeasible action sequence and hence won by Eve. This intuition is caught by \( r2w \) chains defined in Section 2. We now re-state and prove Lemma 16:

(Lemma 16) Fix an arbitrary finite-memory Adam strategy \( \lambda_f^f \) winning in \( G_{r2w}^e \). There is a number \( B \geq 0 \) that bounds the depths of all \( r2w \) chains coming from \( \lambda_f^f \):

\[
\forall x \in R. \forall i \geq 0. \forall r2wch \text{ from } (x,i) : \text{depth}(r2wch) \leq B.
\]

Proof. Recall that with every play in \( G_{r2w}^e \) we can associate the action word and the constraint sequence \( C_0 C_1 \ldots \) (the latter is constructed by constr from Lemma 13).

We prove the lemma by contradiction, by constructing a play with \( \lambda_f^f \) which induces an unsatisfiable constraint sequence and therefore is losing for Adam.

Suppose the lemma does not hold for some finite-memory winning Adam strategy \( \lambda_f^f \). Then the set of constraint sequences induced by the plays with \( \lambda_f^f \) has unbounded-depth \( 2w \) chains. Apply the Ramsey argument from Lemma 2 to this set of constraint sequences: the set induces unbounded-depth \( 1w \) chains. The set of these \( 1w \) chains contains at least one of the two: unbounded-depth \( 1w \) chains that are (i) increasing, (ii) decreasing.

Consider the first case, the other one is similar. The number of registers \( R \), constraints over \( R \), and states in \( \lambda_f^f \) is finite (since \( \lambda_f^f \) is finite-memory), but these \( 1w \) increasing chains have unbounded depth (and hence length). Therefore there exist a \( 1w \) increasing chain \( \chi \) and moments \( m \) and \( n \) such that \( C_m = C_n \), the chain \( \chi \) goes through the same register \( r \) in \( m \) and \( n \), and Adam strategy \( \lambda_f^f \) is in the same state \( q \) in both moments. Moreover, as the chain has unbounded depth, we can assume that the chain segment \( \sigma \) from \( m \) to \( n \) has at least one strict increase, so its depth \( > 0 \). Note that Adam cannot distinguish the moments \( m \) and \( n \), so if Eve repeats her actions between \( m \) and \( n \), Adam will respond in the same way. Hence the constraints from \( m \) to \( n \) will be repeated too, and the chain segment \( \sigma \) will be extended to \( \sigma \cdot \sigma \). By making Eve repeat her actions from \( m \) to \( n \) forever, we can construct a play consistent with \( \lambda_f^f \) that has a constraint sequence with an infinite increasing \( 1w \) chain \( \chi' \). The chain eventually repeats the segment \( \sigma \) forever, and since the depth of \( \sigma \) is nonzero, the chain has infinite depth. But we cannot derive a contradiction yet by proclaiming that the play \( \pi \) is losing by Adam, because having an infinite increasing \( 1w \) chain, per se, does not make the constraint sequence unsatisfiable. We need one more step.

Recall that the \( 1w \) chain \( \chi \), from which we started in the previous paragraph, was constructed using the Ramsey argument from some \( r2w \) chain. As a consequence, \( \chi \) consists solely of the elements of the original \( r2w \) chain. Note that in \( r2w \) chains the starting element is one of the largest elements (by definition, \( 2w \) chains are decreasing). Therefore, all elements of the \( 1w \) chain \( \chi \) are also nonstrictly smaller than \( (x,i) \), where \( (x,i) \) is the starting element of the \( r2w \) chain; the same holds for the elements of the infinite \( 1w \) increasing chain \( \chi' \). But this requires an infinite number of values between \( \nu_i(x) \) and \( 0 \), which is impossible in \( \mathbb{N} \), so our constructed constraint sequence is unsatisfiable. Hence the constructed play \( \pi \) with \( \lambda_f^f \) is losing for Adam, which contradicts the assumption of winning \( \lambda_f^f \), concluding the proof. ◀
Church Synthesis on Register Automata over Linearly Ordered Data Domains

Proof of Lemma 24 (Adam data strategy)

The lemma essentially follows from Lemmas 13, 16, and 9.

**Proof of Lemma 24.** Fix an Adam strategy \( \lambda^f_A : V_2(V_3)^* \rightarrow V_3 \) (equiv., \( \lambda_A : \Sigma^* \rightarrow \text{Tst} \)) winning in \( G_f^{r_\text{eq}} \). From \( \lambda^f_A \), we construct \( \lambda^3_A : \Sigma^* \rightarrow \mathbb{N} \) step-by-step as follows.

Suppose we are in the middle of a play: \( \delta_0...\delta_{k-1} \) has been played by Adam \( \lambda^3_A \) and \( \sigma_0...\sigma_{k-1} \) by Eve; both sequences are empty initially. We want to know the value \( \delta_k \) for \( \lambda^3_A(\sigma_0...\sigma_{k-1}) \). Let \((v_0, v_1), (u_0, u_1)(v_1, v_2)...(u_k, v_k)\) be the current play prefix of the register game (initially \((v_0, v_0)\)). We construct the corresponding play prefix \( \nu_0, \nu_1, \nu_2...\nu_k \) of \( G_f \) (initially \( v_0 \)). We assume that this play prefix adheres to \( \lambda^f_A \) (this holds initially). Let \((\text{tst}_k, \text{asgn}_k, u_k) = \lambda_A(\sigma_{k-1}, v_k) \) be the next vertex chosen by Adam in \( G_f \).

Let \( C_0...C_{k-1} \) be the constraint sequence built from \((\text{tst}_0, \text{asgn}_0)...(\text{tst}_k, \text{asgn}_k)\) by the mapping \( \text{constr} \) of Lemma 13. Recall that all \( C_i \) are over registers \( R \cup \{r_d\} \), where \( r_d \) is a fresh register whose role is to store the last Adam data value. By Lemma 16, the strategy \( \lambda_A \) induces constraint sequences over \( R \cup \{r_d\} \) whose \( r_2w \) chains are depth-bounded by \( B \).

Hence we can apply the data-assignment function from the proof of Lemma 9 (page 26). This gives \( \nu_{k+1} \) and in particular the value for \( r_d \) at moment \( k + 1 \), which is the last Adam value, so we set \( \delta_k = \nu_{k+1}(r_d) \). The specification automaton evolves into the configuration \((u_k, v_{k+1})\), whereas \( G_f \) evolves into \((\text{tst}_k, \text{asgn}_k, u_k)\). Thus, the extended play adheres to the winning Adam strategy \( \lambda^f_A \).

Therefore, starting from empty sequences of Adam and Eve actions, step-by-step we construct the values for \( \lambda^3_A \). Since the strategy \( \lambda^3_A \) adheres to \( \lambda^f_A \), it is winning in \( G \).

B.6 Definitions and Proofs of Section 4

**Input-driven register automata**

An input-driven deterministic register automaton is a two-sided register automaton whose output data are required to be the content of some registers. Formally, it is a tuple \( S = (Q, q_0, R, \delta, \alpha) \) where \( Q = Q_A \uplus Q_E \), \( q_0 \in Q_A \) and the transition function is

\[
\delta : (Q_A \times \text{Tst}) \rightarrow \text{Asgn} \times Q_E \cup (Q_E \times \text{Tst}) \rightarrow \text{Asgn}_{\varnothing} \times Q_A,
\]

where \( \text{Tst}_{\text{eq}} \) consists of tests which contain at least one atom of the form \( * = r \) for some \( r \in R \), i.e. the output data must be equal to some specification register, and \( \text{Asgn}_{\varnothing} = \{ \varnothing \} \) meaning that output data output is never assigned to anything (this is without loss of generality, given that the output data has to be equal to the content of some register).

**Register transducers**

A register transducer (RT) is a tuple \( T = (Q, q_0, R, \delta) \), where \( Q \) is a set of states and \( q_0 \in Q \) is initial, \( R \) is a finite set of registers. The transition function \( \delta \) is a (total) function \( \delta : Q \times \text{Tst} \rightarrow \text{Asgn} \times R \times Q \).

The semantics of \( T \) are provided by the associated register automaton \( A_T \). It has states \( Q' = Q_A \uplus Q_E \), where \( Q_A \) and \( Q_E \) are two disjoint copies of \( Q \), it has initial state \( q_0 \) and set of registers \( R \). Its transition function is defined as \( q \xrightarrow{\text{tst}_0, \text{asgn}_0} q' \) if and only if \( q \xrightarrow{T, \text{tst}_1, \text{asgn}_1} q' \) stands for \( \delta(q, \text{tst}_1) = (\text{asgn}_1, r, q') \) (similarly for \( A_T \). The priority function is simply \( \alpha : q \rightarrow 2 \), i.e. all states are accepting. Then, \( T \) recognises the (total) function \( f_T : d_0^T d_1^T \cdots \rightarrow d_0 d_1^T \cdots \) such that \( d_0^T d_1^T d_2^T \cdots \in L(A_T) \). It exists since all
Proof of Theorem 17

With an input-driven register automaton specification \( S \), we associate a one-sided register automaton \( S' \) by treating output registers as finite labels, and then reduce the synthesis problem to deciding the existence of a winning strategy for Eve in the corresponding Church game.

Let \( S = (Q, q_0, R, \delta, \alpha) \) be a specification. We define \( S' = (\text{Tst}_\omega, Q, q_0, R, \delta', \alpha) \) (note that the finite output alphabet is \( \text{Tst}_\omega \)). First, up to remembering equality relations between registers, we can assume that from an output state, all outgoing transitions are takeable, independently of the register configuration, i.e. that from a reachable output configuration \((q_E, \tau)\), for all transitions \( t = q_E \xrightarrow{\text{Tst}, \tau} q'_A \), there exists \( d \) such that \( q_E \xrightarrow{d} q'_A \). This however induces a blowup of \( |Q| \) exponential in \(|R|\).

The transition function is \( \delta'_A = \delta_A \), and \( \delta'_E(q_E, \text{Tst}) = q'_A \) if and only if \( \delta E(q_E, \text{Tst}) = (\emptyset, q'_A) \). Overall, the size of \( S' \) is exponential in \(|R|\) (because of the assumption we made on output transitions) and polynomial in \(|Q|\).

Now, we need to show that \( S \) admits a register transducer implementation if and only if Eve has a winning strategy in the Church game \( G \) associated with \( S' \).

First, assume that there exists a register transducer \( T \) which realises \( S' \). From \( T \), we define a strategy \( \lambda_T \) in \( G \), which simulates \( T \) and \( S \) in parallel. Given an history \( \delta_0^n \ldots \delta_n \), let \( \delta^{(n)}_i \) be the data output by \( T \). As \( S \) is deterministic, there exists a unique run over the history \( \delta_0^n \ldots \delta_n \). Let \( t = q_E \xrightarrow{\text{Tst}, \tau} q'_A \) be the transition taken by \( S \) on reading \( \delta^n \). Then, define \( \lambda_T(\delta_0^n \ldots \delta_n) = \text{Tst}_\omega \). Now, for a play in \( G \) consistent with \( \lambda_T \), consider the associated run in \( S' \). As \( T \) is an implementation and the sequence of transitions is feasible (as witnessed by the data given as input), such run is necessarily accepting, so \( \lambda_T \) is indeed a winning strategy in \( G \).

Conversely, assume that Eve has a winning strategy in \( G \). By Proposition 15, she has a winning strategy in \( G_f \). We can assume such strategy to be a finite-memory strategy with memory \( M \), initial memory \( m_0 \) and update function \( \mu : M \times V_3 \rightarrow M \). \( \lambda_E : M \rightarrow \text{Tst}_\omega \). Then, consider \( T = (Q \times M, (q_0, m_0), R, \delta') \). We define \( \delta' \) as follows: assume the transducer is in state \((q, m)\). Then, the transducer receives input satisfying some test \( \text{Tst}_\omega \). In \( S \), it corresponds to some input transition \( \delta(q, \text{Tst}) = (\text{asgn}, q') \). The memory is updated to \( \mu(m, (\text{Tst}, \text{asgn})) = m' \), and \( \lambda_E(m') = \text{Tst}_\omega \). Let \( r \) be such that \( \text{Tst}_\omega = r_\infty \) (such \( r \) necessarily exists by definition of \( \text{Tst}_\omega \)). Then, we let \( \delta((q, m), \text{Tst}) = (\text{asgn}, r, (q', m')) \). Now, let \( w = \delta_0^n \delta_1^n \ldots \) be an input data word, and \( T(w) = \delta_0^n \delta_1^n \ldots \). By construction, the run of \( S \) over \( w \otimes T(w) = \delta_0^n \delta_1^n \delta_2^n \ldots \) corresponds to a play consistent with \( \lambda_E \), so it is accepting (since it is feasible, as witnessed by \( w \otimes T(w) \)).

As a consequence, \( w \otimes T(w) \in L(S) \), which means that \( T \) is indeed a register transducer implementation of \( S \).

Overall, we reduced the transducer synthesis problem of \( S \) to solving a Church game over \( S' \), whose size is polynomial in \(|Q|\) and exponential in \(|R|\). By Theorem 11, this yields an algorithm polynomial in \(|Q|\) and exponential in \( c \) and \(|R|\) (the exponentials do not stack).

\textbf{Remark 25.} For data domain \((Q, \leq)\), the synthesis problem for specifications defined by two-sided register automata is also decidable, if the target implementation is any program, as the Church game again reduces to a parity game: checking feasibility is still doable using a parity automaton. However, in general, register transducers might not suffice; e.g. the environment can ask the system to produce an infinite sequence of data in increasing order.
Yet, it can be shown that implementations can be restricted to simple programs, which can be modelled by register transducers which have the additional ability to pick a data between two others, e.g. by computing $\frac{d_1 + d_2}{2}$, and to pick a data above all others, e.g. by multiplying by 2. This limited computational power suffice to translate a finite-memory strategy in the feasibility game to an implementation.