Church Synthesis on Register Automata over Linearly Ordered Data Domains

Léo Exibard
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
Université libre de Bruxelles, Brussels, Belgium

Emmanuel Filiot
Université libre de Bruxelles, Brussels, Belgium

Ayrat Khalimov
Université libre de Bruxelles, Brussels, Belgium

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 ω-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 ω-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 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 ω-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 → Logic and verification; Theory of computation → Automata over infinite objects; Theory of computation → Transducers

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

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.28

Related Version *Full Version*: https://arxiv.org/abs/2004.12141

Funding This work was supported by the Fonds de la Recherche Scientifique - FNRS under Grant n°F.4510.9. Emmanuel Filiot is research associate of the Fonds de la Recherche Scientifique - FNRS.

1 Introduction

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 ω-regular languages. The seminal papers [12, 33] 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 [39, 25, 32], software with integer parameters [10], communication protocols with message parameters [15], and more [9, 37, 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, 27, 28, 17].

Register automata extend finite-state automata to infinite alphabets $\mathcal{D}$ by introducing a finite number of *registers* [26]. 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, 27, 28, 17] focused on register automata and transducers operating in the domain $(\mathcal{D}, =)$ equipped with *equality* tests only. Related works [22, 31] 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 ℓ above the value $\nu(c)$ of the counter, saves it into a register c_{new}, and asks Eve to provide the value between $\nu(c)$ and $\nu(c_{new})$. 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 $r_l < * < r_M$

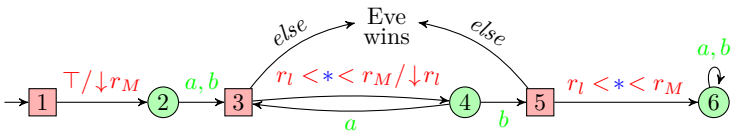

Figure 1 Eve wins this game in $\mathbb{N}$ but loses in $\mathbb{Q}$.

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 $]\mathit{d}_l, \mathit{d}_M[$, 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 $\geq \mathit{d}_M$, losing the game. An alternative Eve winning strategy, that does depend on Adam data, is to loop in 3–4 until $\mathit{d}_M - \mathit{d}_l = 1$ (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 $]\mathit{d}_l, \mathit{d}_M[$ for any $\mathit{d}_l < \mathit{d}_M$, 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 $r_l^1 = r_M^1$, 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 $r_l^2 < r_M^2$ in state 2. Otherwise, if Adam had provided a data equal to the value in r_l, the constraint would be $r_l^2 = r_M^2$. In this way the constraints evolve during the play, forming an infinite sequence. Looping in states 3–4 induces the constraint sequence $\left(r_l^i < r_l^{i+1} < r_M^i = r_M^{i+1}\right)_{i>2}$. It forms an infinite chain $r_l^3 < r_l^4 < \ldots$ bounded by constant $r_M^3 = r_M^4 = \ldots$ 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 [35] which studies, among others, the nonemptiness problem of constraint automata, whose states and transitions are described by constraints. In particular, they show [35, Appendix C] that satisfiability of constraint sequences can be checked by *nondeterministic* ωB-automata [4]. Nondeterminism however poses a challenge in synthesis, and it is not known whether games with winning objectives as nondeterministic ωB-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 two kinds of constraint sequences inspired by Church games with register automata. First, we show that the satisfiable lasso-shaped (regular) constraint

sequences, of the form uv^ω, are recognisable by deterministic *parity* automata. Second, we show how to assign values to registers on-the-fly in order to satisfy a constraint sequence induced by a play in the Church game.

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 results on satisfiability of constraint sequences of two special kinds. 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. [19] 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 [20] studies strategy synthesis but, again, mainly in the dense domain. A similar one-sided setting was studied in [21] for Church games with a winning condition given by logical formulas, but only for $(\mathcal{D}, =)$. The work on automata with atoms [30] 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 [38, 11, 1] or one-counter systems [36, 23] 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, $\mathbb{N} = \{0, 1, \dots\}$. 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 $r \in R$.

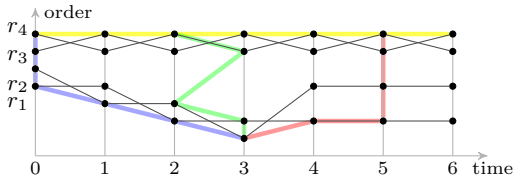

Figure 2 Visualisation of a constraint sequence. Individual register values are depicted by black dots, and dots are connected by black lines when they talk about the same register. Blue/red/-green/yellow paths depict chains.

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 $\mathcal{D}$. In what follows, the symbol $\bowtie$ denotes one of $>$, $<$, or $=$. A *constraint* is a maximal consistent set of atoms of the form $t_1 \bowtie t_2$ 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 $\nu_a \colon \{r_1 \mapsto 1, r_2 \mapsto 4\}$ and $\nu_b \colon \{r_1 \mapsto 2, r_2 \mapsto 3\}$. 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 $r \bowtie s$ for $r, s \in R$, and $C_{|R'}$ – the subset of atoms over primed registers. Given a set S of atoms $r' \bowtie s'$ over $r', s' \in R'$, let $\mathit{unprime}(S)$ be the set of atoms derived by replacing every $r' \in R'$ by r.

A *constraint sequence* is an infinite sequence of constraints $C_0 C_1 \dots$ (when we use finite sequences, we explicitly state it). It is *consistent* if for every i: $\mathit{unprime}(C_{i|R'}) = C_{i+1|R}$, 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 $\nu \in \mathcal{D}^R$, define $\nu' \in \mathcal{D}^{R'}$ to be the valuation that maps $\nu'(r') = \nu(r)$ for every $r \in R$. A valuation $w \in \mathcal{D}^{R \cup R'}$ *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 $\nu_0 \nu_1 \ldots \in (\mathcal{D}^R)^\omega$ 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 $C_0 C_1 \dots$ start with

$$\{r_1 < r_2 < r_3 < r_4, r_4 = r'_3, r_3 = r'_4, r_1 = r'_1, r_1 > r'_2\}\{r_2 < r_1 < r_4 < r_3, r_4 = r'_3, r_3 = r'_4, r_1 = r'_1, r_2 > r'_1\}$$

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 $\mathbb{Q}$ and in $\mathbb{N}$. 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 $\mathbb{N}$. Also, the constraint C_0 requires all registers in R to differ, hence the sequence is not 0-satisfiable in $\mathbb{Q}$ nor in $\mathbb{N}$. Another example is given by the sequence $(\{r > r'\})^\omega$ with $R = \{r\}$: it is satisfiable in $\mathbb{Q}$ but not in $\mathbb{N}$. ⌟

Satisfiability of constraint sequences in $\mathbb{Q}$. The following result is glimpsed in several places (e.g. in [35, Appendix C]): a constraint sequence is satisfiable in $\mathbb{Q}$ iff it is consistent. This is a consequence of the following property which holds because $\mathbb{Q}$ dense: for every

constraint C and $\nu \in \mathbb{Q}^R$ such that $\nu \models C_{|R}$, there exists $\nu' \in \mathbb{Q}^{R'}$ 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 $\mathbb{Q}$ are recognizable by deterministic parity automata (see [18]).

▶ **Theorem 1.** *There is a deterministic parity automaton of size exponential in $|R|$ that accepts exactly all constraint sequences satisfiable in $\mathbb{Q}$. The same holds for 0-satisfiability.*

Satisfiability of constraint sequences in $\mathbb{N}$. Fix R and a constraint sequence $C_0C_1\dots$ over R. A (decreasing) *two-way chain* is a finite or infinite sequence $(r_0, m_0) \triangleright_0 (r_1, m_1) \triangleright_1 \ldots \in \big((R \times \mathbb{N}) \cdot \{=, >\}\big)^{*,\omega}$ 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 $(r_i \triangleright_i r_{i+1}) \in C_{m_i}$.
- If $m_{i+1} = m_i + 1$ then $(r_i \triangleright_i r'_{i+1}) \in C_{m_i}$.
- If $m_{i+1} = m_i - 1$ then $(r_{i+1} \triangleright_i r'_i) \in C_{m_i - 1}$.

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 $(r_4, 2) > (r_3, 3) > (r_2, 2) > (r_1, 3) > (r_2, 3)$ 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 $\triangleright_i$ being the equality $=$; it can also be written as $(m, r_0r_1r_2\ldots)$. Given a stable chain $\chi_r = (m, r_0r_1\ldots)$ and a chain $\chi_s = (s_0, n_0) \bowtie_0 (s_1, n_1) \bowtie_1 \ldots$, such that $n_i \geq m$ for all plausible i, the chain χ_r is *non-strictly above* χ_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 $(m, r_0r_1\ldots)$ is *maximal* if it is non-strictly above all other stable chains starting after m. In Figure 2, the yellow chain $(0, (r_4r_3)^\omega)$ 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') *$\exists \mathrm{B} \in \mathbb{N}$: all trespassing two-way chains have depth $\leq \mathrm{B}$ (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 $\nu_0\nu_1\dots$ 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 [18]. ◀

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 $A \wedge B$ implies $A' \wedge B'$ (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 [18]. ◀

The next lemma proved in [18] refines the characterisation to 0-satisfiability.

▶ **Lemma 4.** *A consistent constraint sequence is* 0*-satisfiable in* $\mathbb{N}$ *iff it satisfies conditions* $A \wedge B$ *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* ≥ 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 $\mathrm{B} \in \mathbb{N}$ 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 ω-regular automata. For instance, they can express the non-ω-regular set of words $w = a^{n_1}ba^{n_2}b\ldots$ such that $n_i \leq \mathrm{B}$ for all $i \geq 0$, for some $\mathrm{B} \in \mathbb{N}$ 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 $>$ 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 χ that currently exhibits a stable behaviour. To every such chain χ, the master automaton assigns a tracer automaton whose task is to ensure the absence of unbounded-depth trespassing chains below χ. For that, it uses $2|R|$ counters and requires them to be bounded. The overall acceptance condition ensures that if the chain χ is stable, then there are no trespassing chains below χ 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 [18].) ◀

▶ Remark. [35, Appendix C] shows that satisfiable constraint sequences in $\mathbb{N}$ are characterised by nondeterministic ωB-automata [4], which are strictly more expressive than max-automata.

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

Lasso-shaped sequences (ω-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 [18]):

▶ **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 ω-regular automaton:

▶ **Theorem 8.** *For every* R*, there is a deterministic parity automaton that accepts a lasso-shaped constraint sequence iff it is* 0*-satisfiable in* $\mathbb{N}$*; its number of states is exp. 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 $(x, i) \triangleright (r_1, m_1) \triangleright (r_2, m_2) \triangleright \dots$ 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 $\nu \in \mathcal{D}^R, \nu' \in \mathcal{D}^{R'}$ s.t. $\nu \cup \nu' \models C_i$: $\left|\{r' \in R' \mid \forall s \in R.\, \nu'(r') \neq \nu(s)\}\right| \leq 1$ (*assumption* †). Intuitively: at most one new value can appear (but many disappear) during the step of the constraint (see also [18]). 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* $\mathrm{B} \geq 0$*, there exists a data-assignment function* $f : (\mathsf{C}_{|R} \cup \mathsf{C}^+) \to \mathbb{N}^R$ *such that for every finite or infinite meaningful constraint sequence* $C_0C_1C_2\dots$ *satisfying assumption* † *and whose r2w chains are depth-bounded by* B*, the register valuations* $f(C_{0|R})f(C_0)f(C_0C_1)\dots$ *satisfy the constraint sequence.*

Proof idea. We define a special kind of $xy^{(m)}$-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 $xy^{(m)}$-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 $\mathit{@}_r$ and $\mathit{@}_s$, then set $\mathit{@}_x = \lfloor \frac{\mathit{@}_r + \mathit{@}_s}{2} \rfloor$; and if the constraint puts a register x above all other registers, then set $\mathit{@}_x = \mathit{@}_M + 2^{\mathrm{B}}$ where $\mathit{@}_M$ the largest value currently held in the registers and B is the given bound on the depth of r2w chains. Full proof is in [18]. ◀

3 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 λ_A and λ_E, the *outcome* $\lambda_A \| \lambda_E$ is the infinite sequence $i_0 o_0 i_1 o_1 ...$ such that for all $j \geq 0$: $i_j = \lambda_A(o_0 ... o_{j-1})$ and $o_j = \lambda_E(i_0 ... i_j)$. 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 λ_E such that for every Adam strategy λ_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 ω-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 $* \bowtie r$ for $r \in R$ and $\bowtie \in \{=, <, >\}$. 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 $(r_1 < *) \wedge (* < r_2)$ is a test. We denote Tst_R the set of all tests and just Tst if R is clear from the context. A register valuation $\nu \in \mathcal{D}^R$ and data $\mathit{d} \in \mathcal{D}$ *satisfy* a test $\mathsf{tst} \in \mathsf{Tst}$, written $(\nu, \mathit{d}) \models \mathsf{tst}$, if all atoms of tst get satisfied when we replace the placeholder $*$ by d and every register $r \in R$ by $\nu(r)$. An *assignment* is a subset $\mathsf{asgn} \subseteq R$. Given an assignment asgn, a data $\mathit{d} \in \mathcal{D}$, and a valuation ν, we define $\mathit{update}(\nu, \mathit{d}, \mathsf{asgn})$ to be the valuation ν' s.t. $\forall r \in \mathsf{asgn} \colon \nu'(r) = \mathit{d}$ and $\forall r \notin \mathsf{asgn} \colon \nu'(r) = \nu(r)$.

A *deterministic register automaton* is a tuple $S = (Q, q_0, R, \delta, \alpha)$ where $Q = Q_A \uplus Q_E$ 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*, $\delta = \delta_A \uplus \delta_E$ is a (total and deterministic) *transition function* $\delta_P : (Q_P \times \mathsf{Tst} \to \mathsf{Asgn} \times Q_{P'})$ for $P \in \{A, E\}$ and the other player P', and $\alpha : Q \to \{1, ..., c\}$ is a *priority function* where c is the *priority index*.

A *configuration* of A is a pair $(q, \nu) \in Q \times \mathcal{D}^R$, describing the state and register content; the *initial configuration* is $(q_0, 0^R)$. A *run* of S on a word $w = \mathit{d}_0 \mathit{d}_1 ... \in \mathcal{D}^\omega$ 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 tst_i be a unique test for which $(\nu_i, \mathit{d}_i) \models \mathsf{tst}_i$, we have $\delta(q_i, \mathsf{tst}_i) = (\mathsf{asgn}_i, q_{i+1})$ for some asgn_i and $\nu_{i+1} = \mathit{update}(\nu_i, \mathit{d}_i, \mathsf{asgn}_i)$. Because the transition function δ is deterministic and total, every word induces a unique run in S. The run ρ 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 $c < \mathit{d} < c'$ (the formal proof can be found in [18]).

▶ **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 [21] for domain $(\mathcal{D}, =)$). Specifications are now given as a language $S \subseteq (\mathcal{D} \cdot \Sigma)^\omega$. 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 Σ of Eve *labels*, and its transition function $\delta = \delta_A \uplus \delta_E$ now has $\delta_E : Q_E \times \Sigma \to Q_A$ while $\delta_A : Q_A \times \mathsf{Tst} \to \mathsf{Asgn} \times Q_E$ stays as before. Runs on words in $(\mathcal{D} \cdot \Sigma)^\omega$ are defined as before except that register valuations are updated only in Adam states. We omit the formal definitions. Figure 1 shows an example of a one-sided automaton. For instance, it rejects the words $3a1b2(\Sigma\mathcal{D})^\omega$ and accepts the words $3a1a2b(\mathcal{D}\Sigma)^\omega$.

▶ **Theorem 11.** *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.*

The proof of the theorem relies on the notion of action words. An *action word* is a sequence $(\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1)...$ from $(\mathsf{Tst} \times \mathsf{Asgn})^{*,\omega}$. An action word is $\mathcal{D}$*-feasible* if there exists a sequence $\nu_0 \ell_0 \nu_1 \ell_1 \dots$ of register valuations ν_i and data ℓ_i over $\mathcal{D}$ such that $\nu_0 = 0^R$ and for all plausible i: $\nu_{i+1} = \mathit{update}(\nu_i, \ell_i, \mathsf{asgn}_i)$ and $(\nu_i, \ell_i) \models \mathsf{tst}_i$. We first outline the proof structure and then provide the details.

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 $\Leftarrow$ is easy, because Eve winning strategy λ_E^f in G_f, which picks finite labels in Σ depending on the history of transitions of S, can be used to construct Eve winning strategy $\lambda_E : \mathbb{Q}^+ \to \Sigma$ in G by simulating the automaton S. To prove the other direction, we assume that Adam has a winning strategy λ_A^f in G_f, which picks tests depending on the history of transitions of S, then construct an Adam *data* strategy $\lambda_A : \Sigma^* \to \mathbb{Q}$ that concretises these tests into data values. This data instantiation is easy because $\mathbb{Q}$ 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 ω-regular subset of feasible words sufficient for our purpose, and correspondingly define an ω-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 λ_A^f, we construct Adam data strategy $\lambda_A : \Sigma^* \to \mathbb{N}$, relying on the finite-memoriness of the strategy λ_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 $G = (V_\forall, V_\exists, v_0, E, W)$ where $V_\forall$ and $V_\exists$ are disjoint finite sets of *vertices* controlled by Adam and Eve, $v_0 \in V_\forall$ is *initial*, $E \subseteq (V_\forall \times V_\exists) \cup (V_\exists \times V_\forall)$ is a turn-based *transition relation*, and $W \subseteq (V_\forall \cup V_\exists)^\omega$ is a *winning objective*. An *Eve strategy* is a mapping $\lambda : (V_\forall \cdot V_\exists)^+ \to V_\forall$ such that $(v_\exists, \lambda(v_0 ... v_\exists)) \in E$ for all paths $v_0 ... v_\exists$ of G starting in v_0 and ending in $v_\exists \in V_\exists$. 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 $\pi = v_0 v_1 \dots$ is *compatible* with an Eve strategy λ when for all $i \geq 0$ s.t. $v_i \in V_\exists$: $v_{i+1} = \lambda(v_0 \dots v_i)$. 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 [24] (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 $G_f = (V_\forall, V_\exists, v_0, E, W_f)$. Essentially, it memorises the transitions taken by the automaton S during the play of Adam and Eve. It has $V_\forall = \{q_0\} \cup (\Sigma \times Q_A)$, $V_\exists = \mathsf{Tst} \times \mathsf{Asgn} \times Q_E$, $v_0 = q_0$, $E = E_0 \cup E_\forall \cup E_\exists$ where:

- $E_0 = \big\{\big(v_0, (\mathsf{tst}, \mathsf{asgn}, u_0)\big) \mid \delta(v_0, \mathsf{tst}) = (\mathsf{asgn}, u_0)\big\}$,
- $E_\forall = \big\{\big((\sigma, v), (\mathsf{tst}, \mathsf{asgn}, u)\big) \mid \delta(v, \mathsf{tst}) = (\mathsf{asgn}, u)\big\}$, and
- $E_\exists = \big\{\big((\mathsf{tst}, \mathsf{asgn}, u), (\sigma, v)\big) \mid \delta(u, \sigma) = v\big\}$.

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

$$W_f = \big\{v_0(\mathsf{tst}_0, \mathsf{asgn}_0, u_0)(\sigma_0, v_1) \dots \mid (\mathsf{tst}_0\mathsf{asgn}_0) \dots \in \mathsf{Feasible}_\mathcal{D}(R) \Rightarrow v_0 u_0 v_1 u_1 \dots \models \alpha\big\}$$

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 $\mathit{unprime}(C_{|R'})$ are state constraints.

Every action word naturally induces a unique constraint sequence. For instance, for registers $R = \{r, s\}$, an action word starting with $(\{r < *, s < *\}, \{s\})$ (test whether the current data ℓ 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 [18].

▶ **Lemma 13.** *Let R be a set of registers, $R_d = R \uplus \{r_d\}$, and $\mathcal{D} \in \{\mathbb{N}, \mathbb{Q}\}$. There exists a mapping $\mathit{constr} : \Pi \times \mathsf{Tst} \times \mathsf{Asgn} \to \mathsf{C}$ from state constraints Π over R_d and tests-assignments over R to constraints C over R_d, such that for all action words $a_0 a_1 a_2 ... \in (\mathsf{Tst} \times \mathsf{Asgn})^\omega$, $a_0 a_1 a_2 ...$ is feasible iff $C_0 C_1 C_2 ...$ is 0-satisfiable, where $\forall i \geq 0$: $C_i = \mathit{constr}(\pi_i, a_i)$, $\pi_{i+1} = \mathit{unprime}(C_{i|R'_d})$, $\pi_0 = \{r = s \mid r, s \in R_d\}$.*

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 $\mathcal{D} = \mathbb{Q}$ and as a deterministic max-automaton for $\mathcal{D} = \mathbb{N}$. As a consequence of Theorem 1 (resp. 5), we get (see full proof in [18]):

▶ **Lemma 14.** *W_f is definable by a deterministic parity automaton if $\mathcal{D} = \mathbb{Q}$ and a deterministic max-automaton if $\mathcal{D} = \mathbb{N}$. Moreover, these automata are polynomial in $|Q|$ and exponential in $|R|$, and for $\mathcal{D} = \mathbb{Q}$, 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 [18].

The main goal is to show that Eve wins G iff she wins G_f. The direction $\Leftarrow$ is easy: Eve has less information in G_f, as she only has access to the tests satisfied by the input data, so she is stronger in G. Conversely, assume by contraposition that Eve does not win G_f. As ω-regular games are determined, Adam has a winning strategy λ_A^f in G_f. It induces a strategy λ_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 λ_A^f is winning, such run is accepting, so λ_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 1 of the theorem. Item 2 (determinacy) and Remark 12 are then a consequence of the finite-memory determinacy of ω-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 [18].

Using ω-regular game G_f^{reg} instead of G_f. W_f is not ω-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 ω-regular subset $W_f^{reg} \subseteq W_f$ which is equi-realisable to W_f. Let $\mathsf{QFeasible}_{\mathbb{N}}(R)$ be the set of *quasi-feasible* action words over R, defined as the set of words $\overline{a}$ such that its induced constraint sequence (through the mapping *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 ≥ 1 from moment 0; by Lemma 7, this entails 0-satisfiability of lasso-shaped constraint sequences. We then let:

$$W_f^{reg} = \left\{ v_0(\mathsf{tst}_0, \mathsf{asgn}_0, u_0)(\sigma_0, v_1)\ldots \mid (\mathsf{tst}_0, \mathsf{asgn}_0)\ldots \in \mathsf{QFeasible}_{\mathbb{N}}(R) \Rightarrow v_0u_0v_1u_1\ldots \models \alpha \right\}$$

From Lemma 8, we can build a deterministic parity automaton with a number of states exponential in $|R|$ and polynomial in $|Q|$ and a priority index linear in c recognising W_f^{reg}. Let G_f^{reg} be the finite-arena game with the same arena as G_f, with winning condition W_f^{reg}. We now show that the Church game G reduces to G_f^{reg} (full proof in [18]).

▶ **Proposition 15.** *Eve has a winning strategy in G iff she has a winning strategy in G_f^{reg}.*

Proof idea. If Eve has a winning strategy in G_f^{reg}, then, since $\mathsf{Feasible}_{\mathbb{N}}(R) \subseteq \mathsf{QFeasible}_{\mathbb{N}}(R)$, we have that $W_f^{reg} \subseteq W_f$, so it is also winning in G_f. Now, the argument for $\mathbb{Q}$ 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^{reg}. As ω-regular games are finite-memory determined, Adam has a finite-memory winning strategy λ_A^f in G_f^{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 so-called 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* $\text{B} \geq 0$ *that bounds the depths of all r2w chains coming from* λ_A^f*: for all constraint sequences resulting from playing with* λ_A^f*, for all* $x \in R$*, for all* $i \geq 0$*, we have that for all r2wch from* (x, i)*,* $depth(r2wch) \leq \text{B}$*.*

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 λ_A^f which induces an unsatisfiable constraint sequence and therefore is losing for Adam. Assume that the constraint sequences induced by the plays with λ_A^f have unbounded-depth 2w chains. By Ramsey argument from Lemma 2, the constraint sequences have unbounded-depth *1w* chains. Along those chains, as λ_A^f is finite-memory, there is a repeating configuration with 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 λ_E^f 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 λ_A^f is winning. ◀

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

Since G_f^{reg} 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 [18].

▶ **Theorem 17.** *For specifications defined by deterministic input-driven output register automata over data domains $\mathbb{Q}$ and $\mathbb{N}$, 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_f^{reg}. If Eve wins G_f^{reg}, 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 $(\mathbb{N}, \leq)$ are decidable, in 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 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, 29], 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, 34] 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 [21] 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 ω-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 Mikołaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic*, pages 41–55, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

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

8 Mikołaj 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 R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 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 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over infinite ordered alphabets, 2020. `arXiv:2004.12141`.

19 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.

20 Azadeh Farzan and Zachary Kincaid. Strategy synthesis for linear arithmetic games. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–30, 2017.

21 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`.

22 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.

23 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.

24 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.

25 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.

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

27 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.

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

29 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. doi:10.4230/LIPIcs.CONCUR.2019.25.

30 Bartek Klin and Mateusz Łełyk. Scalar and Vectorial mu-calculus with Atoms. *Logical Methods in Computer Science*, Volume 15, Issue 4, October 2019. doi:10.23638/LMCS-15(4:5)2019.

31 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.

32 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.

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

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

35 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.

36 Olivier Serre. Parity games played on transition graphs of one-counter processes. In *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-31, 2006, Proceedings*, pages 337–351, 2006.

37 V. Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT '09*, pages 1–13, 2009.

38 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.

39 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.