Register Games on Infinite Ordered Data Domains

Léo Exibard
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Emmanuel Filiot
Université libre de Bruxelles, Belgium

Ayrat Khalimov
Université libre de Bruxelles, Belgium

Abstract

We introduce two-player turn-based zero-sum *register* games on an infinite linearly ordered data domain. Register games have a finite set of registers intended to store data values. At each round, Adam picks some data in the domain, which is tested against the data contained in the registers, using the linear order. Depending on which test holds (exactly one is required), Eve decides to assign, or not, the data to some of her registers, and the game deterministically evolves to a successor vertex depending on her assignment choice. Eve wins the game if she has a strategy which depends only on the tests that hold (and not on the concrete data values of Adam), such that whichever values Adam provides, the sequence of visited vertices of the game satisfies some parity condition. We show the decidability of register games over data domains $\mathbb{N}$ and $\mathbb{Q}$. For $\mathbb{Q}$, they can be solved in ExpTime and finite-memory strategies always suffice to win. For $\mathbb{N}$, we show that deciding the existence of a finite-memory strategy is also in ExpTime. We apply these results to solve the synthesis problem of strategies resolving non-determinism in (non-deterministic) register transducers on data words.

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

Keywords and phrases Register Automata, Synthesis, Ordered Data words, Transducers

Digital Object Identifier 10.4230/LIPIcs...

Relationship with tree automata?

1 Introduction

Context. Two-player games on finite graph arenas are a powerful mathematical framework whose main application is the automatic synthesis of reactive systems [26, 7]. In the game metaphor to synthesis, the system to be synthesised is modelled as a player called Eve, the protagonist, while the environment in which the system is executed is the opponent player called Adam. The goal is to decide whether there exists a strategy such that whatever actions Adam takes, the protagonist can react by some actions which, in the long run, guarantee that the sequence of visited vertices of the game arena satisfy some winning condition W. A finite-memory strategy winning for Eve is then a system which is by construction correct with respect to the condition W. Classically, the winning condition is assumed to be ω-regular and given by some ω-automaton (e.g. a deterministic parity automaton), and in this context it is decidable to check whether Eve has a winning strategy [26]. The kind of systems that can be synthesised by solving ω-regular games are limited to simple finite-state machines. There have been many extensions of this classical two-player zero-sum game setting to model more realistic synthesis scenarios: quantitative games can model the synthesis of systems which meet some quantitative constraints [38, 17, 15, 21, 11, 4], games with imperfect information can model the synthesis of systems which have only a partial view of the environment's actions [33, 21, 5, 14, 31], non zero-sum games can model the synthesis of multi-component systems executed in an environment and whose components have their own objectives not necessarily antagonistic [36, 16, 6, 30, 20, 18, 2, 23]. See also [12] for a survey.

Register games. We pursue this line of research by introducing a certain kind of two-player zero-sum games that are played on an *infinite* arena. Our games can model the synthesis of systems that manipulate data from an infinite linearly ordered domain $\mathcal{D}$, such as $\mathbb{N}$ or $\mathbb{Q}$, through the use of data registers and tests over those registers. Given a finite set of registers R, *register games* are played by Adam and Eve in the following way: initially, all

strange

registers contain an initial value (0 for $\mathcal{D} \in \{\mathbb{N}, \mathbb{Q}\}$). Adam picks a data $d \in \mathcal{D}$ which is tested against the registers content. Tests are maximally consistent conjunctions of atoms (also known as types in logics) of the form $* < r$, $r < *$, or $r = *$, where $*$ is a placeholder for the data chosen by Adam. In other words, a test must provide full information about how the data compares to the contents of *all* registers. Since tests are maximally consistent, only one test can be true for any given data and valuation of registers. Depending on which test holds (and not on the concrete data Adam chooses), Eve decides to store the data in some of her registers, or none. Moreover, game arenas have a finite number of vertices whose dynamics is deterministic with respect to the tests and assignments. A play is an infinite sequence of states resulting from Adam-Eve interaction; it is won by Eve if it satisfies some parity condition. Eve wins the game if she has a strategy which, with any sequence of

no condition on registers!

successful tests, associates some assignment of her registers, such that all plays compatible with her strategy are winning. Since Eve's strategies are test-based and do not depend on the concrete data picked by Adam, we can finitely represent a register game by allowing Adam to directly choose a test rather than a concrete value, as illustrated in the following example.

Example. Figure 1 illustrates a register game arena where Adam states are squares and Eve's states are circles. Eve's objective is to eventually reach state 7, while Adam tries to avoid it. There are three registers r_M, r_c, r_n. Tests depicted on the arrows are not maximally consistent: they are macros expressing all maximally consistent tests implied by them. For example, the test $\top$ (true) stands for all maximally consistent tests, and the test $r_c < * < r_M$ stands for all maximally consistent tests ϕ such that $\phi \to r_c < * < r_M$, such as $\phi = r_c < * < r_M \wedge * < r_n$. Initially, Adam provides some data d_M, meant to be

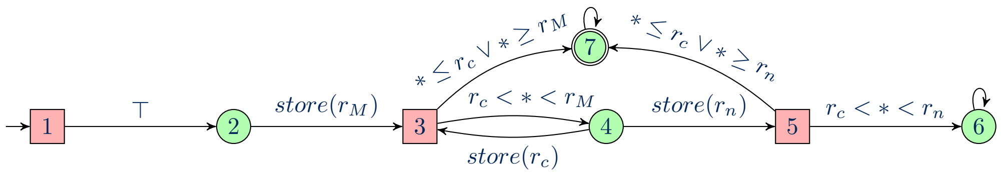


Figure 1 Register game in which Eve has a winning strategy in $\mathbb{N}$ but not in $\mathbb{Q}$.

an upper bound, and which is stored in r_M by Eve. Register r_c, initially at 0, is meant to contain the last data $\mathscr{d}_c$ played by Adam. From state 3, he can either provide some data outside of the interval $]\mathscr{d}_c; \mathscr{d}_M[$, losing immediately (transition to state 7), or provide some data strictly between $\mathscr{d}_c$ and $\mathscr{d}_M$. Then, Eve can either store the data in r_n, and the game deterministically evolves to state 5, or update r_c and move to state 3. She cannot do the latter forever, because her goal is to visit 7. From state 5, if Adam can provide a data strictly between the previous one (in r_c) and the current one (in r_n), he wins. Otherwise he provides a data outside of $]\mathscr{d}_c; \mathscr{d}_n[$ and loses.

Eve has a winning strategy in $\mathbb{N}$, but not in $\mathbb{Q}$: in $\mathbb{N}$, her strategy is to always store the current data into r_c (transition from 4 to 3). Eventually, after a finite number of steps, Adam will provide a data that is above $\mathscr{d}_M$ and the game will go to state 7, which is winning for Eve. In $\mathbb{Q}$, however, Adam can provide infinitely many data below the initial threshold, e.g. by setting $\mathscr{d}_M = 1$ and then giving the sequence of values $(1 - \frac{1}{n})_{n \in \mathbb{N}}$. If, at some point, Eve stores data in r_n, Adam is able to provide a data between r_c and r_n, since $\mathbb{Q}$ is dense.

Synthesis of register transducers. A motivation behind this work is automatic synthesis of reactive systems from specifications. In the classical approach to synthesis, there are two finite alphabets of input symbols Σ_I and output symbols Σ_O. A specification S is an ω-regular set of ω-words in $(\Sigma_I \cdot \Sigma_O)^\omega$, and the goal is to check the existence of a strategy $\lambda : \Sigma_I^* \to \Sigma_O$ which guarantees that the words $w \in (\Sigma_I \cdot \Sigma_O)^\omega$ generated by this strategy, whatever symbols are input, all belong to S. When S is given as a formula in monadic second-order logic, this problem is decidable and finite-memory strategies, which can be represented as deterministic finite transducers (i.e., Mealy machines) always suffice to realise the specification [13]. Deterministic finite transducers extend deterministic finite automata with outputs: they alternately read one input symbol and produce one output symbol. When S is given as a deterministic parity automaton A, solving the synthesis problem reduces to solving a parity game played on A, seen as an arena. Solving this game amounts to finding a strategy which, given the current output state (a state reading symbols in Σ_O) plus possibly some additional memory, selects the next output symbol (and therefore the next transition as A is deterministic) so that in the long run, the parity condition is satisfied. In the special case of parity games, it is known that memoryless strategies suffice.

Recently, this classical synthesis setting has been generalised to infinite data domains, ? e.g. $\mathbb{N}$: inputs and outputs are taken in $\Sigma_I \times \mathbb{N}$ and $\Sigma_O \times \mathbb{N}$ respectively. In this context, specifications are given by universal register automata [29, 28, 22], which can test data for equality only. Strategies take the form of deterministic register transducers which are also restricted to testing data for equality. It was shown that given $k \in \mathbb{N}$ and a specification S given as a universal register automaton, it is decidable whether there exists a register transducer with k registers that realises the specification [29, 22]. In this paper, we are motivated by extending those results to specifications and strategies which can test their data with respect to a linear order. Register games are a powerful tool in this context.

Contributions. Our main contribution is to show that register games are decidable for $\mathbb{N}$ and $\mathbb{Q}$ (Theorem 1). For $\mathbb{Q}$, it is decidable in ExpTime and finite-memory strategies always suffice ~~to~~ for Eve to win. For $\mathbb{N}$, we leave open the complexity in the general case but we prove that deciding whether Eve has a *finite-memory* winning strategy is in ExpTime. As an application, we prove the decidability of a synthesis problem of deterministic register transducers over linearly ordered data domains. More precisely, specifications are given as (non-deterministic) register transducers of the following form: whenever they read an input data, they can test it against the register contents (using possibly the linear-order), and can then assign the data and output the content of some of their registers, called an output register. Multiple output registers and assignments are possible and the goal is to test the existence of a finite-memory strategy which resolves this non-determinism, i.e. selects a single output register and a single assignment depending on the input data that has been provided. We show that this problem is decidable in ExpTime for $\mathbb{N}$ and $\mathbb{Q}$ (Theorem 16), as a consequence of the ExpTime solvability of register games with finite-memory. By taking the product of such a finite-memory strategy with the register transducer defining the specification, we obtain a *deterministic* register transducer realising the specification.

To prove the decidability of register games, we rely on a thorough study of what we call (infinite) constraint sequences, which is interesting in its own and has a dedicated section (Section 3). Infinite constraint sequences talk about the evolution over time of a finite set of variables (which correspond to the registers of the register game). Coming back to Example 1, looping in states 3/4 induces the infinite constraint sequence $(r_c^{(i)} < r_c^{(i+1)} \wedge r_c^{(i)} < r_M^{(i)})_{i \geq 0}$, where $r_c^{(i)}$, resp. $r_M^{(i)}$, denotes the value of r_c, resp. r_M, at step i. Clearly, it is not satisfiable in $\mathbb{N}$ and therefore in playing the game, Eve can use this information to win. We prove that constraint sequences satisfiable in $\mathbb{Q}$ are ω-regular, while in $\mathbb{N}$ they are definable by deterministic max-automata [8]. Using this characterisation, we show that register games over $\mathbb{Q}$ (resp. over $\mathbb{N}$) reduce to two-player zero-sum games with an ω-regular objective (resp. with a winning objective given as a deterministic max-automaton). The latter games are decidable as a consequence of a result of [9]. We further push our study of constraint sequences and prove that, even though satisfiable constraint sequences in $\mathbb{N}$ are not ω-regular, with respect to winning register games with finite-memory strategies, we can always consider an ω-regular subset of them which preserves the fact that Eve wins the game with a finite-memory strategy.

obvious

interesting

Related works. Games on infinite arenas induced by pushdown automata [32, 37, 10, 1] or one-counter systems [35, 25] are orthogonal to register games. The synthesis of strategies resolving non-determinism is a standard problem in automata theory, which has been considered for ω-regular automata [27], weighted automata [3, 24] and counter-automata [19].

A characterisation of satisfiable constraint sequences in $\mathbb{N}$ similar to ours was given in [34], which greatly inspired our work. In [34], the authors prove that constraint sequences satisfiable in $\mathbb{N}$ are recognisable by non-deterministic ωB-automata, while we prove they are recognisable by deterministic max-automata. First, non-deterministic ωB-automata are strictly more expressive than deterministic max-automata. Second, two-player games with an objective given as a non-deterministic ωB-automaton are *not* known to be decidable. It is not clear how to get our characterisation by deterministic max-automata from the proof of [34]. We use slightly different notions which simplify the proof of our characterisation: while the authors of [34] express properties of *sets* of chains (chains are monotonic subsequence of constraints of a constraint sequence), we express properties of *single* chains. Finally, we show that with respect to solving register games with finite-memory strategies, it is not necessary to consider all satisfiable constraint sequences in $\mathbb{N}$ but only an ω-regular subset of them. It is not clear how to infer such a result from [34].

2 Register Games

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 $\leq$. We also distinguish a special element of $\mathcal{D}$ denoted 0 (its choice is not important). In this paper, we consider two data domains: $\mathbb{N}$ or $\mathbb{Q}$ ordered with the natural order $\leq$. Let $\mathcal{D}$ be some data domain. *Register games* are two-player zero sum games played on a finitely presented infinite-state game arena. The opponent, Adam, provides a data in $\mathcal{D}$ by the form of tests over a finite set of registers $R = \{r_1, \dots, r_k\}$ intended to store data in $\mathcal{D}$. So, instead of giving a concrete data, Adam indicates its position in the linear order w.r.t the registers, e.g. the test $r_1 < * < r_2$ indicates that the data is between r_1 and r_2 ($*$ is a placeholder for the concrete data). The protagonist, Eve, can store this data into some registers, and hence her actions are modelled as a subset of registers in which to put the data. Depending on these actions, the game evolves in different vertices, equipped with a parity condition to define the winning plays.

Registers, assignments and tests. Let $\mathcal{D}$ be some data domain. We let $R = \{r_1, \dots, r_k\}$ be a set of elements called *registers*, intended to contain data values, i.e. values in $\mathcal{D}$. Given a set of registers R, a *register valuation* is a mapping $\nu : R \to \mathcal{D}$. We denote by Val_R the set of register valuations and ν_0^R (or just ν_0) the constant valuation defined by $\nu_0(r) = 0$ for all $r \in R$. An assignment is a subset $\mathsf{asgn} \subseteq R$. Given an assignment asgn, a data $d \in \mathcal{D}$ and some valuation ν, we define $update(\nu, d, \mathsf{asgn})$ to be the valuation ν' s.t. $\forall r \in \mathsf{asgn}\colon \nu'(r) = d$ and $\forall r \notin \mathsf{asgn}\colon \nu'(r) = \nu(r)$.

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 $R = \{r_1, r_2\}$, $r_1 < *$ 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 $d \in \mathcal{D}$ *satisfy* a test $\mathsf{tst} \in \mathsf{Tst}$, written $(\nu, d) \models \mathsf{tst}$, if all atoms of tst get satisfied when we replace in them the placeholder $*$ by d and every register $r \in R$ by $\nu(r)$.

Register games. A register game is a tuple $G = (R, V, V_\forall, V_\exists, v_0, \Delta, \alpha)$ where R is a finite set of registers, $V = V_\forall \uplus V_\exists$ is a finite set of vertices, Δ is a transition function defined as $\Delta = \Delta_\forall \cup \Delta_\exists$ where $\Delta_\forall : V_\forall \times \mathsf{Tst}_R \to V_\exists$ and $\Delta_\exists : V_\exists \times \mathsf{Asgn}_R \to V_\forall$. Finally, $\alpha : V \to \{0, \dots, n\}$ is a priority function, where n is called the index.

A finite play in G is defined as a finite path of G starting in v_0. The set of finite plays is denoted by $Plays_G$. A play π is an infinite path in G starting in v_0. We say that π satisfies the parity condition α if the maximal priority visited infinitely often is even, i.e. $\max\{\alpha(v_i) \mid v_j = v_i \text{ for infinitely many } i\}$ is even. A strategy for Eve is a mapping $\lambda : \mathsf{Tst}_R^+ \to \mathsf{Asgn}_R$. A *finite memory* strategy for Eve is a strategy which can be represented by a finite-state machine $M = (Q, q_0, T)$ such that Q is a finite set of states with initial state q_0, and $T : Q \times \mathsf{Tst}_R \to \mathsf{Asgn}_R \times Q$ is a (total) transition function. The machine M defines the strategy λ_M by $\lambda_M(\mathsf{tst}_1 \dots \mathsf{tst}_n) = T(q, \mathsf{tst}_n)$ where q is the state reached by M after reading $\mathsf{tst}_1 \dots \mathsf{tst}_{n-1}$ from state q_0.

We now define the notion of winning strategy in G, which reflects the infinite-state nature of register games and is parameterised by a data domain $\mathcal{D}$. An *action word* $\overline{a} = \mathsf{tst}_0 \mathsf{asgn}_0 \dots$ is a finite or infinite sequence in $(\mathsf{Tst}_R \cdot \mathsf{Asgn}_R)^\omega \cup (\mathsf{Tst}_R \cdot \mathsf{Asgn}_R)^*$. We say that $\overline{a}$ is a *labelling* of a play (finite or infinite) $\pi = v_0 u_0 v_1 u_1 \dots$ if for all $i \geq 0$, $u_i = \Delta(v_i, \mathsf{tst}_i)$ and $v_{i+1} = \Delta(u_i, \mathsf{asgn}_i)$. Note that for all action words $\overline{a}$, there is a unique play denoted $\pi_{\overline{a}}$ such that $\overline{a}$ is a labelling of $\pi_{\overline{a}}$. The action word $\overline{a}$ is said to be $\mathcal{D}$*-feasible* (or just feasible) if there exists an infinite sequence $\nu_0 d_0 \nu_1 d_1 \dots$ of register valuations ν_i and data d_i over $\mathcal{D}$ such that $\nu_0 = \nu_0^R$ (the constant valuation of any $r \in R$ to 0) and for all $i \geq 0$,

$\nu_{i+1} = \mathit{update}(\nu_i, d_i, \mathsf{asgn}_i)$ and $\nu_i, d_i \models \mathsf{tst}_i$. An action word $\mathsf{tst}_0\mathsf{asgn}_0\ldots$ is compatible with a strategy λ if for all $i \geq 0$, $\mathsf{asgn}_i = \lambda(\mathsf{tst}_0 \ldots \mathsf{tst}_i)$. The set of action words compatible with λ is denoted by $\mathit{Outcome}_G(\lambda)$. Finally, we define a strategy λ for Eve to be $\mathcal{D}$-winning if for any $\mathcal{D}$-feasible $\overline{a} \in \mathit{Outcome}_G(\lambda)$, $\pi_{\overline{a}}$ satisfies the parity condition. Note that we do not require that all outcomes of λ are feasible. It is because Adam can input tests which are not satisfiable by any concrete data, and these actions should always be winning for Eve. For example, in a register valuation where $\nu(r_1) = \nu(r_2)$, Adam could input the test $r_1 < * < r_2$ which is not satisfiable in the context of ν, as it does not correspond to any concrete data value that Adam could give.

Though not necessary in the paper, the interested reader can find in Appendix a semantics of register games as infinite-state parity games. Here is our main result:

▶ **Theorem 1.** *Given a register game G,*
1. *it is decidable in* ExpTime *whether Eve has a $\mathbb{Q}$-winning strategy in G. Moreover, if she has a $\mathbb{Q}$-winning strategy, then she has $\mathbb{Q}$-winning finite-memory strategy,*
2. *it is decidable whether Eve has an $\mathbb{N}$-winning strategy in G,*
3. *it is decidable in* ExpTime *whether Eve has an $\mathbb{N}$-winning finite-memory strategy in G.*

Sketch of proof – full proof in Section 4. We discuss the case of $\mathbb{N}$, the case of $\mathbb{Q}$ is much easier and left out here. The winning condition of register games asks that any $\mathbb{N}$-feasible action word compatible with the strategy induces a sequence of vertices satisfying the parity condition. The main idea is then to reduce register games G to (classical) two-player zero-sum games G_f, with a winning condition which expresses that either the action word is not $\mathbb{N}$-feasible, or the parity condition is satisfied. More precisely, we reduce register games to games over a finite graph arena (whose transitions are not labelled). Actions taken by Adam and Eve are stored in the vertices of the game. Therefore, if $V = V_\forall \uplus V_\exists$ is the set of vertices of G, the set of Adam's vertices in G_f is $v_0 \cup (V_\forall \times \mathsf{Asgn}_R)$ (the last action taken by Eve is stored but since Adam starts, initially the game is in v_0 the initial vertex of G). Likewise, Eve's vertices in G_f is the set $V_\exists \times \mathsf{Tst}_R$. A play $\pi = v_0(v_1, \mathsf{tst}_1)(v_2, \mathsf{asgn}_2)\ldots$ is winning in G_f if either $\overline{a} = \mathsf{tst}_1\mathsf{asgn}_2\ldots$ is not $\mathbb{N}$-feasible, or $v_0v_1\ldots$ satisfies the parity condition of G. We show that the set of $\mathbb{N}$-feasible actions words is recognisable by a deterministic max-automaton (Lemma 12 in Section 3), and since they are closed under complement and union with an ω-regular language, we can conclude for decidability, since games with a winning condition defined by a deterministic max-automaton are decidable (Theorem 9), based on a result of [9]. To obtain ExpTime for checking the existence of $\mathbb{N}$-winning finite-memory strategies for Eve, we show that it is sufficient to consider an ω-regular winning condition instead of a deterministic max-automata condition (Section 4). Those results are based on a study of satisfiable constraint sequences (Section 3). ◀

3 Satisfiability of Constraint Sequences

The section does not depend on the preceding definitions and can be read independently.

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. We also 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, at the end, and between each other. 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 for instance satisfied 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 constraint $\{r_1 < r_2, r_1 > r'_1, r_2 < r'_2, r'_1 > r'_2\}$ is not satisfiable.

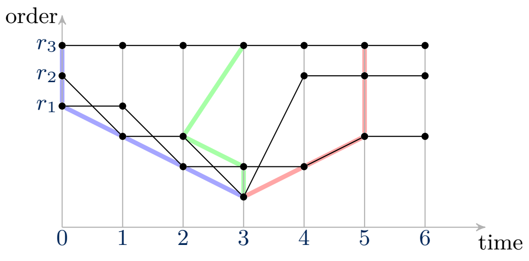


Figure 2 Visualisation of a constraint sequence. After time 6, the values stay the same. Individual register values are depicted by black dots, and dots are connected by black lines when they talk about the same register. Black paths depict threads, and blue/red/green paths depict chains.

Given a constraint C, let $C_{|R}$ denote the subset of its atoms $r \bowtie s$ for $r, s \in R$, and $C_{|R'}$ — the subset of its atoms $r' \bowtie s'$ for $r', s' \in R'$. 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 $C_0 C_1 \dots$. It is *consistent* if, for every i, $\mathit{unprime}(C_{i|R'}) = C_{i+1|R}$ (the register order at the end of step i is equal to the register order at the beginning of step $i+1$). Given a valuation $\mathfrak{v} \in \mathcal{D}^R$, define $\mathfrak{v}' \in \mathcal{D}^{R'}$ to be the valuation that maps $\mathfrak{v}'(r') = \mathfrak{v}(r)$ for every $r \in R$. A valuation $\mathfrak{w} \in \mathcal{D}^{R \cup R'}$ *satisfies* a constraint C, written $\mathfrak{w} \models C$, if every atom is true in $\mathcal{D}$ when every $r \in R \cup R'$ is replaced by $\mathfrak{w}(r)$. A constraint sequence is *satisfiable* if there exists a sequence of valuations $\mathfrak{v}_0 \mathfrak{v}_1 \dots \in (\mathcal{D}^R)^\omega$ such that $\mathfrak{v}_i \cup \mathfrak{v}'_{i+1} \models C_i$ for all $i \geq 0$. If, additionally, $\mathfrak{v}_0 = 0^R$ for some element $0 \in \mathcal{D}$, then it is called 0-*satisfiable*. Notice that satisfiability implies consistency.

Examples. Let $R = \{r_1, r_2, r_3\}$. Let a consistent constraint sequence $C_0 C_1 \dots$ start with

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

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 $r'_2 < r'_1 < r'_3$), 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 (e.g. r_3 does not change over time). The constraint C_0 describes the transition from moment 0 to 1, and C_1, from 1 to 2.

The sequence of Figure 2, where after step 6 the registers do not change, is satisfiable in $\mathbb{Q}$ and in $\mathbb{N}$. For example, the valuations can start with $\mathfrak{v}_0 = \{r_3 \mapsto 5, r_2 \mapsto 4, r_1 \mapsto 3\}$. But no valuations with $\mathfrak{v}_0(r_3) < 5$ 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 $\mathbb{N}$, because any natural number can only be decreased finitely many times.

Satisfiability of constraint sequences in $\mathbb{Q}$. We show that a constraint sequence is satisfiable in $\mathbb{Q}$ iff it is consistent. It is a consequence of the following property (true because $\mathbb{Q}$ is dense):for every constraint C and $\mathfrak{v} \in \mathbb{Q}^R$ such that $\mathfrak{v} \models C_{|R}$, there exists $\mathfrak{v}' \in \mathbb{Q}^{R'}$ such that $\mathfrak{v} \cup \mathfrak{v}' \models C$. Being consistent is a local property to be tested on any two consecutive constraints of the sequence, it is not difficult to show that consistent constraint sequences (and hence constraint sequences satisfiable in $\mathbb{Q}$) are recognizable by deterministic parity automata (shown in Appendix):

▶ **Theorem 2.** *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 $C_0 C_1 \dots$ over R. For $r \in R$, an r-*thread* is a projection of $C_0 C_1 \dots$ into the atoms $r \bowtie r'$ where $\bowtie \in \{>, <, =\}$; thus it is a sequence of atoms $(r \bowtie_0 r')(r \bowtie_1 r') \dots$. An r-thread *stabilises* if it is of the form $u \cdot (r = r')^\omega$. Among stabilised threads with registers $R_s \subseteq R$, there is a *maximal* r_m-thread: it satisfies $\exists i. \forall j > i. \forall r \in R_s \colon (r_m > r) \in C_j \vee (r_m = r) \in C_j$. In the constraint sequence in Figure 2, r_3 gets stabilised; it is maximal when r_1 and r_2 do not stabilise or stabilise below r_3.

A thread describes a history of changes of some fixed register. In contrast, a chain (defined below) relates values of possibly different registers at consecutive moments.

A (decreasing) *two-sided chain* is a finite or infinite sequence $(r_0, m_0) \rhd_0 (r_1, m_1) \rhd_1 \ldots \in \big((R \times \mathbb{N}) \cdot \{=,>\}\big)^{*,\omega}$ (where m_0 does not have to be 0) satisfying the following.

- $m_{i+1} = m_i$ (time freezes) 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 $(r_i \rhd_i r_{i+1}) \in C_{m_i}$.
- If $m_{i+1} = m_i + 1$ then $(r_i \rhd_i r'_{i+1}) \in C_{m_i}$.
- If $m_{i+1} = m_i - 1$ then $(r_{i+1} \rhd_i r'_i) \in C_{m_i - 1}$.

The *depth* of a chain is the number of $>$; when it is infinity, the chain is *infinitely* decreasing. Figure 2 shows three two-sided chains. In blue color, we have a chain $(0, r_3) > (0, r_2) > (0, r_1) > (1, r_2) > (2, r_1) > (3, r_2)$ of depth 5. Similarly, we define *one-sided* chains to be either increasing or decreasing, with forwards-flowing time (thus, m_{i+1} equals m_i or $m_i + 1$). In Figure 2, the blue chain is one-sided decreasing, the red chain is one-sided increasing.

Given a stabilising r-thread, a (two-sided) chain $(m_0, r_0) \rhd_0 (m_1, r_1) \rhd_1 \ldots$ is *lower* than the r-thread if for every (m_i, r_i), either $(r > r_i) \in C_{m_i}$ or $(r = r_i) \in C_{m_i}$. Given the set of all stabilising threads, *trespassing chains* are all the chains lower than the maximal stabilising r_m-thread. The number of trespassing chains in a constraint sequence can be infinite; it can also be zero, e.g. when there are no stabilising threads.

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

(A') *it has no infinitely decreasing two-sided chains; and*

(B') $\exists B \in \mathbb{N}$*: all trespassing two-sided chains have depth at most* B *(we say they have* bounded *depth).*

Sketch of proof – full proof in Appendix. 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 the maximal stabilising chain, which is impossible. For the other direction, we show that if A and B hold, then one can construct a sequence of valuations $\nu_0\nu_1 \ldots$ satisfying the constraint sequence, such that for all $r \in R$, $\nu_i(r)$ is the largest depth of a (decreasing) two-sided chain starting in r at moment i. ◀

We can strengthen the previous lemma to talk only about one-sided chains.

▶ **Lemma 4.** *A consistent constraint sequence is satisfiable in* $\mathbb{N}$ *iff*

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

(B) *the trespassing (increasing or decreasing) one-sided chains have a bounded depth.*

Sketch of proof – full proof in Appendix. Thanks to Lemma 3, we show that $A \wedge B$ implies $A' \wedge B'$ (the other direction is trivial). Let us prove $\neg A' \Rightarrow \neg A$. From an infinite (decreasing) two-sided chain, we can always extract an infinite decreasing one-sided chain, since two-sided 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 $\neg B' \implies \neg B$. Given a sequence of trespassing two-sided chains of unbounded depth, we are able to construct a sequence of one-sided chains of unbounded depth. This construction is more difficult than for showing $\neg A' \implies \neg A$. Indeed, even though there are by hypothesis deeper and deeper trespassing two-sided 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-sided chain and extract from it an arbitrarily deep one-sided chain. However, we show, using a Ramsey argument, that it is still possible to extract arbitrarily deep one-sided chains as the two-sided chains are not completely independent. ◀

The next lemma shown in Appendix refines the previous characterisation to 0-satisfiability.

▶ **Lemma 5.** *A consistent constraint sequence is 0-satisfiable in $\mathbb{N}$ iff it*

- *satisfies conditions $A \wedge B$ from Lemma 4,*
- *starts in C_0 s.t. $C_{0|R} = \{r = s \mid r, s \in R\}$, and*
- *has no decreasing one-sided chains of depth ≥ 1 from $(r, 0)$ for any r.*

We are now able to provide the main result about recognisability of satisfiable constraint sequences by automata. To state the following theorem, we need the notion of *max-automata* [8]. These automata are standard automata (over a finite alphabet) extended with a finite set of counters $c_1, \ldots, c_n$ which can be incremented only, 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 bounded conditions of the form "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. We refer the reader to [8] for a more detailed definition. For instance, the set of words of the form $w = a^{n_1} b a^{n_2} b \ldots$ such that $n_i \leq B$ for all $i \geq 0$, for some $B \in \mathbb{N}$ that depends on w only, is definable by a deterministic max-automaton but is not ω-regular.

▶ **Theorem 6.** *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 $2|R|$. The same holds for 0-satisfiability in $\mathbb{N}$.*

Sketch of proof – full proof in Appendix. We treat the case of satisfiability. Based on Lemma 4, we design a deterministic max-automaton to check conditions A and B. Condition A can be checked with a deterministic parity automaton which tracks infinite decreasing one-sided chains. For condition B, for each register r, we have a deterministic parity automaton checking whether there is an r-stabilising thread. Then, for each r, we construct a deterministic max-automaton which checks whether all r-trespassing one-sided (decreasing or increasing) chains have bounded depth. For that, one needs counters with a bounded condition. This automaton can be made deterministic by using the max operation to merge information about different one-sided chains that end up in the same register. Finally, we take a product of all these automata. The product preserves determinism. ◀

The next result will come handy later when dealing with finite-memory strategies, so we state it here. We say an infinite sequence is *lasso-shaped* if it is of the form $w = uv^\omega$.

▶ **Lemma 7.** *Suppose a consistent constraint sequence is lasso-shaped, has no trespassing infinitely decreasing nor increasing one-sided chains. If it has no infinitely decreasing one-sided chains (not necessarily trespassing), then it is satisfiable. If, additionally, it has no decreasing one-sided chains of depth ≥ 1 from moment 0 and starts with C_0 s.t. $C_{0|R} = \{r = s \mid r, s \in R\}$, then it is 0-satisfiable.*

Sketch of proof – full proof in Appendix. Suppose a chain satisfies the conditions of the lemma and assume it has no infinitely decreasing one-sided chain. Then, assume that it has unbounded-depth trespassing decreasing one-sided chains. Since it is lasso-shaped, there is a decreasing chain of depth at least the length of the lasso, and so we can show that there is an infinite decreasing one-sided chain, which is a contradiction. Assume is has unbounded-depth trespassing increasing one-sided chains. Similarly, since the constraint sequence is lasso-shaped, one can show that it has an infinite trespassing increasing one-sided chain, which contradicts our assumption. So, such a constraint sequence satisfies conditions (A) and (B) of Lem. 4, hence it is satisfiable. The 0-satisfiability case is shown similarly. ◀

Since the previous lemma does not talk about boundedness of chains, one do not need counters anymore, so the conditions of Lemma 7 can be checked by an ω-regular automaton:

▶ **Lemma 8.** *For every R, there is a deterministic parity automaton of size exponential in $|R|$ that accepts exactly all consistent constraint sequences that have no trespassing infinitely increasing nor decreasing one-sided chains, and no infinitely decreasing one-sided chains. The same result holds if we additionally require the absence of decreasing one-sided chains of depth ≥ 1 from moment 0, and the start with C_0 s.t. $C_{0|R} = \{r = s \mid r, s \in R\}$.*

4 Solving Register Games (Proof of Theorem 1)

To solve register games, we show how to reduce them, in the case of a data domain $\mathcal{D} \in \{\mathbb{N}, \mathbb{Q}\}$, to a two-player zero-sum turn-based game on a finite arena, with a winning objective which is (1) ω-regular in the case $\mathcal{D} = \mathbb{Q}$, (2) definable by a deterministic max-automaton in the case $\mathcal{D} = \mathbb{N}$ [8] (hence beyond ω-regularity). In case (1), it is well-known that such games are decidable, and for case (2), it is also known to be decidable as a consequence of a result from [9]. While the latter result yields decidability, it does not provide our ExpTime upper bound for the solvability of register games over $\mathbb{N}$ by finite-memory strategies. We then further refine our reduction and show that the latter problem reduces to an ω-regular games. Let us first recall the notion of two-player zero-sum games over finite arenas.

Two-player zero-sum games over finite arenas. A *two-player zero-sum game* (or just two-player game) is a tuple $G = (V, V_\forall, V_\exists, v_0, E, W)$ where $V = V_\forall \uplus V_\exists$ is a finite set of vertices partitioned into vertices controlled by Adam and Eve, $v_0 \in V_\forall$ is the initial vertex, $E \subseteq V_\forall \times V_\exists \cup V_\exists \times V_\forall$ is a turn-based transition relation, and $W \subseteq V^\omega$ is called the *winning objective*. As for register games, a play is an infinite sequence of vertices starting in v_0 and compatible with E. It is winning if it belongs to W. A strategy for Eve is a mapping λ defined on all plays $\pi.v$ where $v \in V_\exists$, such that $\lambda(\pi.v) \in V_\exists$ and $(v, \lambda(\pi.v)) \in E$. A play $\pi = v_0 v_1 \dots$ is compatible with λ if for all $i \geq 0$, if $v_i \in V_\exists$, then $v_{i+1} = \lambda(v_0 \dots v_i)$. A strategy is winning if all plays compatible with it are winning.

It is well-known that games with a winning objective given as a deterministic parity automaton (on infinite words) can be solved in n^d, where n is the size of the game plus the size of the automaton, and d is the index of the parity function [26]. Solving games with a winning objective given as deterministic max-automata (see page 9) is decidable as well:

▶ **Theorem 9** ([9])**.** *The following problem is decidable: given a 2-player game with a winning objective given as a det. max-automaton, check whether Eve has a winning strategy.*

Proof. This result is not directly expressed as such in [9], where it is proved (in Example 2) that two-player games with a winning condition expressed in weak MSO plus the unbounded quantifier U over infinite words (WMSO+U) are decidable, as an application of the decidability of an MSO logic for trees. We do not define WMSO+U here, as it is sufficient to know that WMSO+U can define all languages recognisable by deterministic max-automata [8]. ◀

Reduction to two-player games. We now show how to reduce register games over $\mathcal{D}$ to two-player games over finite arenas. Fix a register game $G = (R, V, V_\forall, V_\exists, v_0, \Delta, \alpha)$. Let

$\mathsf{Feasible}_{\mathcal{D}}(R)$ denote the set of action words over R feasible in $\mathcal{D}$. We construct the two-player game $G_f = (V', V'_\forall, V'_\exists, v'_0, E, W_G)$ where $V' = V'_\forall \uplus V'_\exists$ over a finite arena as follows. Intuitively, G_f memorises in its states the last action taken. Formally, $V'_\forall = \{v_0\} \cup (V_\forall \times \mathsf{Asgn})$, $V'_\exists = V_\exists \times \mathsf{Tst}$, $v'_0 = v_0$, $E = E_0 \cup E_\forall \cup E_\exists$ where $E_0 = \{(v_0, (v_1, \mathsf{tst})) \mid \Delta(v_0, \mathsf{tst}) = v_1\}$, $E_\forall = \{((v, \mathsf{asgn}), (v', \mathsf{tst})) \mid \Delta(v, \mathsf{tst}) = v'\}$ and $E_\exists = \{((v, \mathsf{tst}), (v', \mathsf{asgn})) \mid \Delta(v, \mathsf{asgn}) = v'\}$. The winning condition $W_G \subseteq (V')^\omega$ of G_f is given as the set of words

$$W_G = \big\{v_0(v_1, \mathsf{tst}_1)(v_2, \mathsf{asgn}_2)\ldots \mid \mathsf{tst}_1\mathsf{asgn}_2\ldots \in \mathsf{Feasible}_{\mathcal{D}}(R) \Rightarrow v_0v_1\ldots \models \alpha\big\}.$$

The next lemma, whose proof is in Appendix, shows the correctness of this construction:

▶ **Lemma 10.** *Eve wins G iff Eve wins G_f. Moreover, she wins G with a finite-memory strategy iff she wins G_f with a finite-memory strategy.*

Our objective now is to characterise the set of action words feasible in $\mathbb{Q}$ and $\mathbb{N}$, in order to express the winning condition W_G as a deterministic parity automaton and as a deterministic max-automaton respectively. Any action word naturally induces a unique constraint sequence as defined in Sec. 3. E.g., over two registers $R = \{r, s\}$, any action word starting with $\{r < * \wedge s < *\}.\{s\}$ (test whether the current data d is above r and s and store it in s) induces a constraint sequence starting with $\{r = s \wedge r = r' \wedge s < s' \wedge r' < s'\}$ (the atom $r = s$ is due to all registers being all equal initially in a register game). The next lemma shown in Appendix formalises this intuition (in which for technical reasons we need an additional register to always store the current data):

▶ **Lemma 11.** *Let R be a set of registers, $r_d \notin R$ and $\mathcal{D} \in \{\mathbb{N}, \mathbb{Q}\}$. There exists a mapping $constr$ from action words over R to constraint sequences over $R \cup \{r_d\}$ such that for all action words $\overline{a}$, $\overline{a}$ is feasible in $\mathcal{D}$ iff $constr(\overline{a})$ is 0-satisfiable in $\mathcal{D}$.*

Based on Lemma 11, Theorem 2 (0-satisfiability in $\mathbb{Q}$) and Theorem 6 (0-satisfiability in $\mathbb{N}$) of Sec. 3, we obtain the following result which characterises the sets of feasible action words.

▶ **Lemma 12.** *For every R, the set of all feasible action words $\mathsf{Feasible}_{\mathcal{D}}(R)$ is definable by*
- *a deterministic parity automaton if $\mathcal{D} = \mathbb{Q}$,*
- *a deterministic max-automaton if $\mathcal{D} = \mathbb{N}$.*

Moreover, these automata are exponential in R.

Since parity automata and max-automata are closed under Boolean operations and deterministic max-automata can express all ω-regular languages [8], we get:

▶ **Corollary 13.** *For every register game G, the set W_G can be defined as:*
- *a deterministic parity automaton A if $\mathcal{D} = \mathbb{Q}$,*
- *a deterministic max-automaton B if $\mathcal{D} = \mathbb{N}$.*

Moreover, these automata are exponential in the size of G, and for $\mathcal{D} = \mathbb{Q}$, the index of the priority function of A is the same as for G.

Proof of Theorem 1.(1,2). Corollary 13, Lemma 10 and Theorem 9 yields the decidability of register games for $\mathcal{D} = \mathbb{N}$. For $\mathcal{D} = \mathbb{Q}$, we also get the claimed ExpTime complexity, moreover it is well-known that finite-memory strategies suffice to win 2-player ω-regular games, hence we can conclude the proof for $\mathcal{D} = \mathbb{Q}$ thanks again to Lemma 10.

For $\mathcal{D} = \mathbb{N}$, the result of [9] used to show that two-player games with a winning condition given by a deterministic max-automaton are decidable, does not allow us to conclude that finite-memory strategy suffice (there might not exist regular trees satisfying an WMSO+U formula), and we leave here this question open. We now study the problem of deciding the existence of finite-memory strategies in register games for $\mathbb{N}$.

Reduction to two-player games with an ω-regular winning condition for $\mathcal{D} = \mathbb{N}$. We now prove that for solving the game G_f with winning condition W_G and finite-memory, it suffices to consider an ω-regular subset $W_G^{reg} \subseteq W_G$ which satisfies that Eve wins G_f with winning condition W_G and a finite-memory iff she wins G_f with winning condition W_G^{reg}. We let $\mathsf{QFeasible}_{\mathbb{N}}(R)$ 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* defined in Lemma 11) satisfies the conditions of Lemma 7 which entail 0-satisfiability of lasso-shaped constraint sequences. We then define the winning condition W_G^{reg} as:

$$W_G^{reg} = \{v_0(v_1, \mathsf{tst}_1)(v_2, \mathsf{asgn}_2)\ldots \mid \mathsf{tst}_1\mathsf{asgn}_2\ldots \in \mathsf{QFeasible}_{\mathbb{N}}(R) \Rightarrow v_0v_1\ldots \models \alpha\}.$$

Based on Lemma 8, it is easily shown that W_G^{reg} is ω-regular:

▶ **Lemma 14.** *The set W_G^{reg} can be defined by a deterministic parity automaton with a number of states exponential in R and polynomial in the number of G vertices, and with the same number of priorities as G.*

Finally, the following lemma states that considering W_G^{reg} instead of W_G is sound:

▶ **Lemma 15.** *For all register games G over $\mathcal{D} = \mathbb{N}$, Eve wins G_f with finite-memory and winning condition W_G iff she wins G_f with winning condition W_G^{reg}.*

Proof. Clearly, $W_G^{reg} \subseteq W_G$ since $\mathsf{Feasible}_{\mathbb{N}}(R) \subseteq \mathsf{QFeasible}_{\mathbb{N}}(R)$. Hence, if Eve wins G_f with winning condition W_G^{reg} (which can be assumed to be finite-memory since W_G^{reg} is ω-regular), she also wins G_f with winning condition W_G with the same (finite-memory) strategy. Conversely, assume Eve wins G_f with winning condition W_G and a finite-memory strategy λ but does not win G_f with winning condition W_G^{reg}. Since W_G^{reg} is ω-regular, by determinacy, Adam has a strategy σ, which can be assumed to be finite-memory, such that for all strategies of Eve, the resulting play does not satisfy W_G^{reg}. We exhibit a contradiction. Let π be the play compatible with λ and σ. Since λ is winning, we have $\pi \in W_G^{reg}$ but $\pi \notin W_G$. Therefore, the action word $\overline{a}$ induced by π (by projection) is not feasible but quasi-feasible. Since λ and σ are both finite-memory, π (and $\overline{a}$) are lasso-shaped. By definition of quasi-feasibility, it means that the constraint sequence $constr(\overline{a})$ satisfies the condition of Lemma 7. So, it is 0-satisfiable, and by Lemma 11 we get that $\overline{a}$ is feasible, contradiction. ◀

Proof of Theorem 1.(3). Let G be a register game over $\mathcal{D} = \mathbb{N}$. By Lemma 10, we have that Eve wins G with finite-memory iff she wins G_f with finite-memory and winning condition W_G. Moreover, by Lemma 15, this is equivalent to Eve winning G_f with winning condition W_G^{reg}. By Lemma 14, W_G^{reg} can be represented by a deterministic parity automaton of exponential size with the same number of priorities as G. Hence, G_f with winning condition W_G^{reg} can be solved in ExpTime, concluding the proof. ◀

5 Application to Register Transducer Synthesis

Data words. Let Σ be a finite alphabet of *labels*. A *data word* over Σ and a linearly ordered data domain $\mathcal{D}$ is an infinite sequence of pairs in $\Sigma \times \mathcal{D}$, and we denote by $(\Sigma \times \mathcal{D})^\omega$ the set of data words over Σ and $\mathcal{D}$. For all $i \geq 1$, we denote by $w[i]$ the ith letter of w.

Register transducers. A *register transducer* (RT) is a tuple $T = (\Sigma_I, \Sigma_O, Q, q_0, R, \delta, \alpha)$, where Σ_I and Σ_O are *input* and *output alphabets* of finite labels, Q is a set of *states* and $q_0 \in Q$ is *initial*, R is a finite set of *registers*, α is a *parity function*. The *transition function* δ is a (total) function $\delta\colon Q \times \Sigma_I \times \mathsf{Tst} \to 2^{\mathsf{Asgn} \times \Sigma_O \times R \times Q}$, which is required to be deterministic when the label in Σ_O, the assignment, and the register $r \in R$ have been chosen. Formally, it satisfies: $(\mathsf{asgn}, \sigma_O, r, q_1), (\mathsf{asgn}, \sigma_O, r, q_2) \in \delta(q, \sigma_I, \mathsf{tst})$ implies $q_1 = q_2$. Hence we may write $q' = \delta(\sigma_I, \mathsf{tst}, \mathsf{asgn}, \sigma_O, r)$ when $(\mathsf{asgn}, \sigma_O, r, q') \in \delta(q, \sigma_I, \mathsf{tst})$.

We now define the notion of run and the language semantics of T over a data domain $\mathcal{D}$. A *configuration* of T is a pair $(q, \mathfrak{v}) \in Q \times \mathcal{D}^R$. The configuration $(q_0, 0^R)$ is called *initial*. An *input word* (i-word) is a sequence from $(\Sigma_I \times \mathcal{D})^\omega$; an *output word* (o-word) is a sequence from $(\Sigma_O \times \mathcal{D})^\omega$; and an *input-output word* (io-word) is a sequence from $((\Sigma_I \times \mathcal{D}) \cdot (\Sigma_O \times \mathcal{D}))^\omega$. Given an i-word w_I and an o-word w_O, we construct an io-word $w_I \otimes w_O = w_I[1]w_O[1]w_I[2]w_O[2]\dots$. A *run* of T on an io-word $w = (\sigma_0^I, d_0^I)(\sigma_0^O, d_0^O)(\sigma_1^I, d_1^I)\dots$ is a sequence of configurations $\rho = (q_0, \mathfrak{v}_0)(q_1, \mathfrak{v}_1)\dots$ starting in the initial configuration and such that for every $i \geq 0$, there are asgn_i and r_i s.t. $q_{i+1} = \delta(\sigma_i^I, \mathsf{tst}, \mathsf{asgn}_i, \sigma_i^O, r_i)$ where tst is the unique test holding for $\mathfrak{v}_i$ and d_i^I, $\mathfrak{v}_{i+1} = \mathit{update}(\mathfrak{v}_i, d_i^I, \mathsf{asgn}_i)$, and $d_i^O = \mathfrak{v}_{i+1}(r_i)$. The i-word $w_I = (\sigma_0^I, d_0^I)(\sigma_1^I, d_1^I)\dots$ is called the input word of ρ, and we also say that ρ is a run of T on w_I. The o-word $(\sigma_0^O, d_0^O)(\sigma_1^O, d_1^O)\dots$ is called the output word of ρ.

The run ρ is called *accepting* if it satisfies the parity condition wrt. α. We say that an io-word w is accepted by T if there exists an accepting run of T on w. The *language* of T, denoted $L(T)$, is the set of io-words accepted by T. The *domain* of T, denoted by $\mathrm{dom}(T)$, is the set of all i-words w_I such that there exists an accepting run of T on w_I.

Synthesis problem. Given an RT T, the synthesis problem asks whether there exists a strategy that resolves non-determinism, i.e., selects outputs, while preserving the parity condition. Let us formalize this notion. An *output-selecting strategy* (o-strategy) for T is a finite-state machine $\lambda = (P, p_0, \tau)$ where P is a finite set of states with initial state p_0 and τ is a total transition function of type $\tau\colon P \times Q \times \Sigma_I \times \mathsf{Tst} \to \mathsf{Asgn} \times \Sigma_O \times R \times P$ such that for all $p \in P$ and $(q, \sigma_I, \mathsf{tst}) \in Q \times \Sigma_I \times \mathsf{Tst}$, if $\tau(p, q, \sigma_I, \mathsf{tst}) = (\mathsf{asgn}, \sigma_O, r, p')$, then there exists $q' \in Q$ such that $(\mathsf{asgn}, \sigma_O, r, q') \in \delta(q, \sigma_I, \mathsf{tst})$. In other words, τ selects some element in $\delta(q, \sigma_I, \mathsf{tst})$. By using λ, we can restrict T to a register transducer denoted $T \otimes \lambda$ for which the transition function always outputs a singleton. Formally, $T \otimes \lambda = (\Sigma_I, \Sigma_O, Q \times P, (q_0, p_0), R, \delta^\lambda, \top)$ where $\top$ is a trivial parity condition (always true) and δ^λ is defined by $\delta^\lambda((q, p), \sigma_O, \mathsf{tst}) = \{(\mathsf{asgn}, \sigma_O, r, (q', p'))\}$ such that $\tau(p, q, \sigma_I, \mathsf{tst}) = (\mathsf{asgn}, \sigma_O, r, p')$ and $(\mathsf{asgn}, \sigma_O, r, q') \in \delta(q, \sigma_I, \mathsf{tst})$.

The *output-selecting strategy synthesis problem* (or just synthesis problem) is the problem of deciding, given an RT T, whether there exists an o-selecting strategy λ s.t. $L(T \otimes \lambda) \subseteq L(T)$; such a strategy is called winning and T is called realisable. If such a strategy exists, the problem asks to provide λ as a finite-state machine. An example is given in Appendix.

Note that since the parity condition of $T \otimes \lambda$ is trivial and the transition functions of T and λ are both total, the input domain of $T \otimes \lambda$ is universal. Therefore, if T does not have a universal domain, it is unrealisable. This is realistic in scenarios where inputs are provided by the environment, so we do not want to restrict them.

▶ **Theorem 16.** *The output-selecting strategy synthesis problem for register transducers over data domain $\mathbb{N}$ (resp. $\mathbb{Q}$) is solvable in* ExpTime.

Sketch of proof – full proof in Appendix. We reduce this problem to solving a register game of polynomial size with finite-memory, which is decidable in ExpTime by Theorem 1. The main difference between a register game and the synthesis problem is that register transducers have finite labels and can output the content of a register. First, output registers are considered as letters from a finite alphabet as their actual content do not matter w.r.t. the synthesis of strategies selecting transitions. Then, we encode finite labels using extra registers and force Adam to provide sufficiently many different data in some initial phase that are stored in those extra registers. ◀

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 S. Almagor, O. Kupferman, and G. Perelli. Synthesis of controllable nash equilibria in quantitative objective game. In *Proc. 27th Int. Joint Conf. on Artificial Intelligence*, pages 35–41, 2018.

3 Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Trans. Algorithms*, 6(2):28:1–28:36, 2010.

4 G. Avni, T. A. Henzinger, and V. Chonev. Infinite-duration bidding games. In *Proc. 28th Int. Conf. on Concurrency Theory*, volume 85 of *LIPIcs*, pages 21:1–21:18, 2017.

5 D. Berwanger, K. Chatterjee, M. De Wulf, L. Doyen, and T. A. Henzinger. Strategy construction for parity games with imperfect information. *Information and Computation*, 208(10):1206–1220, 2010.

6 Dietmar Berwanger. Admissibility in infinite games. In *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, pages 188–199, 2007.

7 R. Bloem, K. Chatterjee, and B. Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking.*, pages 921–962. Springer, 2018.

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

9 Mikolaj Boja'nczyk. 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.

10 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.

11 T. Brihaye, V. Bruyère, J. De Pril, and H. Gimbert. On subgame perfection in quantitative reachability games. *Logical Methods in Computer Science*, 9(1), 2012.

12 Véronique Bruyère. Computer aided synthesis: A game-theoretic approach (survey). In *Developments in Language Theory - 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, pages 3–35, 2017.

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

14 K. Chatterjee and L. Doyen. The complexity of partial-observation parity games. In *Proc. 16th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning*, pages 1–14. Springer, 2010.

15 K. Chatterjee and L. Doyen. Energy parity games. In *Proc. 37th Int. Colloq. on Automata, Languages, and Programming*, pages 599–610, 2010.

16 K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Games with secure equilibria. In *Proc. 19th IEEE Symp. on Logic in Computer Science*, pages 160–169, 2004.

17 K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Quantitative stochastic parity games. pages 121–130, 2004.

18 Krishnendu Chatterjee, Laurent Doyen, Emmanuel Filiot, and Jean-François Raskin. Doomsday equilibria for omega-regular games. *Inf. Comput.*, 254:296–315, 2017.

19 Thomas Colcombet. Forms of determinism for automata (invited talk). In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, pages 1–23, 2012.

20 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 121:1–121:15, 2016.

21 A. Degorre, L. Doyen, R. Gentilini, J. Raskin, and S. Torunczyk. Energy and mean-payoff games with imperfect information. In *Proc. 19th Annual Conf. of the European Association for Computer Science Logic*, pages 260–274, 2010.

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

23 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Rational synthesis under imperfect information. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 422–431, 2018.

24 Emmanuel Filiot, Ismaël Jecker, Nathan Lhote, Guillermo A. Pérez, and Jean-François Raskin. On delay and regret determinization of max-plus automata. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017.

25 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.

26 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.

27 T.A. Henzinger and N. Piterman. Solving games without determinization. In *Proc. 15th Annual Conf. of the European Association for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2006.

28 A. Khalimov and O. Kupferman. Register-bounded synthesis. 2019. Full version from arxiv.

29 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.

30 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. URL: https://doi.org/10.1007/s10472-016-9508-8.

31 S. Nain and M.Y. Vardi. Solving partial-information stochastic parity games. In *Proc. 28th IEEE Symp. on Logic in Computer Science*, pages 341–348. IEEE Computer Society, 2013.

32 N. Piterman and M. Vardi. Global model-checking of infinite-state systems. In *Proc. 16th Int. Conf. on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 387–400. Springer, 2004.

33 J.-F. Raskin, K. Chatterjee, L. Doyen, and T. Henzinger. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3(3), 2007.

34 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.

35 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.

36 M. Ummels. The complexity of Nash equilibria in infinite multiplayer games. In *Proc. 11th Int. Conf. on Foundations of Software Science and Computation Structures*, pages 20–34, 2008.

37 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.

38 U. Zwick and M.S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

A Semantics of register games as infinite-state parity games

We associate with G the infinite-state parity game $G^\infty = (V^\infty, V_\forall^\infty, V_\exists^\infty, v_0^\infty, E^\infty, \alpha^\infty)$ defined by $V_\forall = V_\forall \times Val_R$, $V_\exists = V_\exists \times Val_R \times \mathscr{D}$, $v_0^\infty = (v_0, \mathscr{v}_0)$, and $\alpha^\infty((v,\nu)) = \alpha^\infty((v,\nu,\mathscr{d})) = \alpha(v)$. The transitions are defined as $E^\infty = E_\forall^\infty \cup E_\exists^\infty$ where:

$$\begin{array}{rcl} E_\forall^\infty & = & \{((v,\mathscr{v}),(v',\mathscr{v},\mathscr{d})) \mid v \in V_\forall, \mathscr{v} \in Val_R, \mathscr{d} \in \mathscr{D}, \exists \mathsf{tst} \in \mathsf{Tst}_R, \Delta(v,\mathsf{tst}) = v' \wedge \mathscr{v},\mathscr{d} \models \mathsf{tst}\} \\ E_\exists^\infty & = & \{((v,\mathscr{v},d),(\Delta(v,\mathsf{asgn}), \mathit{update}(\mathscr{v},\mathscr{d},\mathsf{asgn})) \mid v \in V_\exists, \mathscr{v} \in Val_R, \mathscr{d} \in \mathscr{D}, \mathsf{asgn} \in \mathsf{Asgn}_R\} \end{array}$$

The notions of plays are defined as for register games. A finite play

$$\pi = (v_0,\mathscr{v}_0)(u_0,\mathscr{v}_0,\mathscr{d}_0)(v_1,\mathscr{v}_1)(u_1,\mathscr{v}_1,\mathscr{d}_1)\dots$$

induces a sequence of tests $\mathsf{tst}_0\mathsf{tst}_1\dots$ such that for all i, tst_i is the unique test such that $\mathscr{v}_i,\mathscr{d}_i \models \mathsf{tst}_i$. Two finite plays π,π' are said to be equivalent, denoted by $\pi \sim \pi'$ whenever they induce the same sequence of tests, and their projection on V are equal. A strategy for Eve in G^∞ is a mapping $\lambda : Plays_{G^\infty} \to V_\forall$ such that for all $\pi \in Plays_{G^\infty}$, $\pi\lambda(\pi) \in Plays_{G^\infty}$. It is said to be *test-driven* if for all finite plays $h_1 = h_1'(v,\mathscr{v}_1,\mathscr{d}_1)$ and $h_2 = h_2'(v,\mathscr{v}_2,\mathscr{d}_2)$, if $h_1 \sim h_2$, then there exists some assignment $\mathsf{asgn} \subseteq R$ such that $\lambda(h_1) = (u, \mathit{update}(\mathscr{v}_1,\mathscr{d}_1,\mathsf{asgn}))$ and $\lambda(h_2) = (u, \mathit{update}(\mathscr{v}_2,\mathscr{d}_2,\mathsf{asgn}))$. Note that then, $h_1\lambda(h_1) \sim h_2\lambda(h_2)$.

▶ **Proposition 17.** *Eve wins a register game G iff she wins the infinite-state parity game G^∞ with a test-based strategy.*

Proof. Suppose there exists a winning strategy $\lambda : \mathsf{Tst}_R^+ \to \mathsf{Asgn}_R$ for Eve in G. We construct a winning strategy λ^∞ for Eve in G^∞. Let $\pi = (v_0,\mathscr{v}_0)(u_0,\mathscr{v}_0,\mathscr{d}_0)(v_1,\mathscr{v}_1)(u_1,\mathscr{v}_1,\mathscr{d}_1)\dots(u_k,\mathscr{v}_k,\mathscr{d}_k)$ be some finite play in G^∞. This play induces a sequence of tests $\rho = \mathsf{tst}_0\mathsf{tst}_1\dots$ and we let $\mathsf{asgn} = \lambda(\rho)$. Then, $\lambda^\infty(\pi) = (v_{k+1},\mathscr{v}_{k+1})$ such that $\Delta(u_k,\mathsf{asgn}) = v_{k+1}$ and $\mathscr{v}_{k+1} = \mathit{update}(\mathscr{v}_k,\mathscr{d}_k,\mathsf{asgn})$. This strategy λ^∞ is test-driven because the actions of Eve only depends on the tests. It is easy to see that it is winning: let π be an infinite play $\pi = (v_0,\mathscr{v}_0)(u_0,\mathscr{v}_0,\mathscr{d}_0)(v_1,\mathscr{v}_1)\dots$ compatible with λ^∞, this play induces an action word $\overline{a} = \mathsf{tst}_0\mathsf{asgn}_0\dots$ which is feasible by $\mathscr{v}_0\mathscr{d}_0\mathscr{v}_1\mathscr{d}_1\dots$. Hence since λ is winning, $v_0u_0\dots$ satisfies the parity condition, hence π is winning as well by definition of α^∞.

Conversely, let λ^∞ be a winning test-based strategy winning for Eve in G^∞. Since it is test-based, it can be seen as a strategy $\lambda : \mathsf{Tst}_R^+ \to \mathsf{Asgn}_R$, which can be shown to be winning for Eve in G. In particular, given $h = \mathsf{tst}_0\dots\mathsf{tst}_k$, we let π^∞ a play in G^∞ which induces this sequence of tests and which is compatible with λ^∞ (if π^∞ does not exist, then we let $\lambda(h) = \mathsf{asgn}$ for some arbitrary asgn). Then, let $\lambda^\infty(\pi^\infty) = (v,\mathscr{v})$, and let asgn be some maximal assignment (for inclusion) such that $\mathscr{v} = \mathit{update}(\mathscr{v}',\mathscr{d},\mathsf{asgn})$, where $\mathscr{v}'$ is the last valuation of π^∞. We let $\lambda(h) = \mathsf{asgn}$. This assignment, asgn depends on the choice of π^∞, but since λ^∞ is test-based, asgn is unique with respect to h, thus λ is well-defined. To show that λ is winning, it suffices to take a feasible action word compatible with λ. Since it is feasible, this action word is actually induced by a proper play π of G^∞ which additionally is compatible with λ^∞, by definition of λ. This yield that π satisfies the parity condition α^∞ since λ^∞ is winning. Thus, by definition of α^∞, we also get that $\pi_{\overline{a}}$ satisfies α as well. Hence λ is winning. ◀

B Proofs of Section 3

B.1 Proof of Lemma 18

Proof. The directions $\Rightarrow$ for both claims are simple, so we prove only direction $\Leftarrow$.

Consider the first claim, direction $\Leftarrow$. Assume the sequence is consistent. We construct $\mathscr{v}_0\mathscr{v}_1\dots \in (\mathbb{Q}^R)^\omega$ such that $\mathscr{v}_i \cup \mathscr{v}_{i+1}' \models C_i$ for all i. The construction proceeds step-by-step and relies on the following fact (†): for every constraint C and $\nu \in \mathbb{Q}^R$ such that $\nu \models C_{|R}$, there exists $\nu' \in \mathbb{Q}^{R'}$ such that $\nu \cup \nu' \models C$. Then define $\nu_0, \nu_1 \dots$ as follows: start with an arbitrary ν_0 satisfying $\nu_0 \models C_{0|R}$. Given $\nu_i \models C_{i|R}$, let ν_{i+1} be any valuation in $\mathbb{Q}^R$ that satisfies $\nu_i \cup \nu'_{i+1} \models C_i$ (it exists by (†)). Since $\nu_{i+1} \models C_{i|R'}$, and $\mathit{unprime}(C_{i|R'}) = C_{i+1|R}$ 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 ν fixes the values for R, and $\nu \models C_{|R}$. Hence we can uniquely order registers R' and the values $\{\nu(r) \mid r \in R\}$ of R on the $\mathbb{Q}$-line. Since $\mathbb{Q}$ 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 $C_0C_1\dots$ is consistent, then by the first claim, it is satisfiable, hence it has a witnessing valuation $\nu_0\nu_1\dots$. 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 $\nu'_0\nu'_1\dots$ 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 $\mathbb{Q}$ 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. ◀

B.2 Proof of Theorem 2

Proof. The alphabet of the automaton 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 $\mathit{unprime}(C_{1|R'}) = C_{2|R}$. 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 $\mathit{unprime}(C_{1|R'}) = C_{2|R}$. 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 $C_{0|R} = \{r = s \mid r, s \in R\}$. ◀

B.3 Proof of Lemma 3

Proof. Direction $\Rightarrow$. Suppose a constraint sequence C_0C_1... is satisfiable by some valuations $\nu_0\nu_1$.... Assume $\neg A'$: there is an infinite decreasing two-sided chain $\chi = (r_0, m_0)(r_1, m_1)$.... Let $\nu_{m_0}(r_0) = d^\star$ 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 χ 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 $d^\star$ and 0, which is impossible in $\mathbb{N}$. Hence A' must hold. Now assume $\neg B'$: there is a sequence of two-sided trespassing chains of unbounded depth. By definition of trespassing, there is at least one stabilised thread; let an r-thread be the maximal among them. Let $d^\star$ be the stabilised value of the r-thread. By definition of trespassing chains, they lay below the r-thread, and the values of registers in them are bounded by $d^\star$, hence the depths of such chains are bounded by $d^\star$, contradicting the assumption $\neg B'$. Hence B' holds.

Direction $\Leftarrow$. Given a consistent constraint sequence C_0C_1... satisfying A' and B', we construct a sequence of register valuations $\nu_0\nu_1$... 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-sided chains from (r, i); the depth $d(r, i)$ can be 0 but not ∞, 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

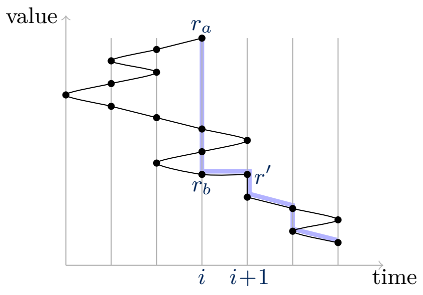


Figure 3 Proving the direction $\neg A' \Rightarrow \neg A$ in Lemma 4. The two-sided chain is in black, the constructed one-sided chain is in blue.

satisfied. Pick an arbitrary atom $t_1 \bowtie t_2$ 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.

- $t_1 \bowtie t_2$ 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 $\mathfrak{v}_i \cup \mathfrak{v}'_{i+1}$ satisfies the atom.
- $t_1 \bowtie t_2$ 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 $\mathfrak{v}_i \cup \mathfrak{v}'_{i+1}$ satisfies the atom.

This concludes the proof. ◀

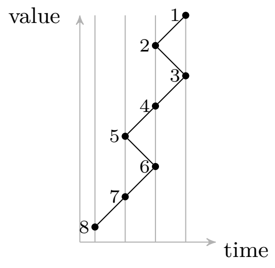
B.4 Proof of Lemma 4

Proof. We show that the conditions $A \wedge B$ hold iff the conditions $A' \wedge B'$ from Lemma 3 hold; which implies the result by Lemma 3. The directions $\neg A \Rightarrow \neg A'$ and $\neg B \Rightarrow \neg B'$ follow from the definition of chains.

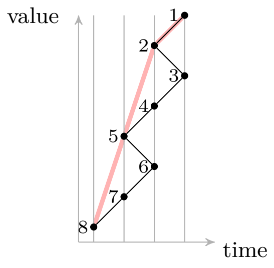
Direction $\neg A' \Rightarrow \neg A$. Given an infinite two-sided chain $\chi = (r_a, i)\ldots$, we construct an infinite descending one-sided chain χ'. The construction is illustrated in Figure 3. Our one-sided chain χ' starts in (r_a, i). The area on the left from i-timeline contains $i \cdot |R|$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

Direction $\neg B' \Rightarrow \neg B$. Given a sequence of trespassing two-sided chains of unbounded depth, we need to create a sequence of trespassing one-sided chains of unbounded depth. We extract a witnessing one-sided chain of a required depth from a sufficiently deep two-sided chain. To this end, we represent the two-sided chain as a clique with colored edges, and whose one-colored subcliques represent all one-sided 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-sided chain.

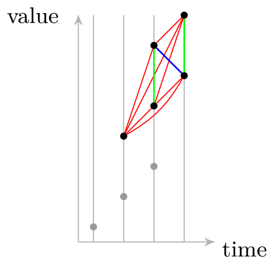
The Ramsey theorem is about clique graphs with colored edges. For the number $n \in \mathbb{N}$ of vertices, let K_n denote the clique graph and E_{K_n} — its edges, and let $\mathit{color}\colon E_{K_n} \to \{1, \ldots, \#c\}$ be the edge-coloring function, where $\#c$ is the number of edge colors in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



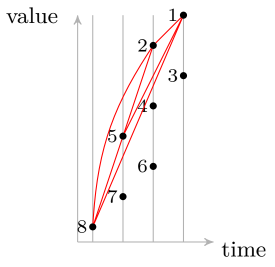
(a) A given two-chain (w/o stuttering)



(d) Constructed increasing one-chain



(b) Clique: shown the edges for the top 5 points only. Try completing the rest.



(c) Monochromatic subclique with elements 1, 2, 5, 8

Figure 4 Proving the direction $\neg B' \Rightarrow \neg B$ in Lemma 4

> Fix the number $\#c$ of edge colors. $(\forall n)(\exists l)(\forall \mathit{color}\colon E_{K_l} \to \{1, \ldots, \#c\})$: 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-sided chains of unbounded depth, we show how to build a sequence of one-sided chains of unbounded depth. Suppose we want to build a one-sided chain of depth n, and let l be Ramsey number for $(3, n)$. Because the two-sided chains from the sequence have unbounded depth, there is a two-sided chain χ of depth l. From it we construct the following colored clique (the construction is illustrated in Figure 4).

- Remove stuttering elements from χ: whenever $(r_i, m_i) = (r_{i+1}, m_{i+1})$ appears in χ, remove (r_{i+1}, m_{i+1}). We repeat this until no stuttering elements appear. Let $\chi_> = (r_1, m_1) > \cdots > (r_l, m_l)$ be the resulting sequence; it is strictly decreasing, and contains l pairs (the same as the depth of the original χ). Note the following property $(\dagger)$: for every not necessarily adjacent $(r_i, m_i) > (r_j, m_j)$, there is a one-sided 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 $\mathit{color}\big((r_a, m_a), (r_b, m_b)\big)$ 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 $V \subseteq \{(r_1, m_1), \ldots, (r_l, m_l)\}$. 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 $\chi^\star = (r_1^\star, m_1^\star) < \cdots < (r_n^\star, m_n^\star)$, where $m_i^\star < m_{i+1}^\star$ and $(r_i^\star, m_i^\star) \in V$, for every plausible i. The sequence $\chi^\star$ may not satisfy the definition of one-sided 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 $(\dagger)$—to construct the one-sided chain $\chi^{\star\star}$ of depth n from $\chi^{\star}$ 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-sided 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 $\neg B' \Rightarrow \neg B$. ◀

B.5 Proof of Theorem 6

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 $\Rightarrow$. The first two items follow from the definition of satisfiability and Lemma 4. 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 $A \wedge B$ hold, the sequence is satisfiable, hence it also satisfies the conditions $A' \wedge B'$ from Lemma 3. In the proof of Lemma 3, we showed that in this case the following valuations $\nu_0\nu_1\ldots$ satisfy the sequence: for every $r \in R$ and moment $i \in \mathbb{N}$, set $\nu_i(r)$ (the value of r at moment i) to the largest depth of the two-sided chains starting in (r, i). We construct $\nu_0\nu_1\ldots$ as above, and get a witness of satisfaction of our constraint sequence. But note that at moment 0, $\nu_0 = 0^R$, by the last item. Hence the constraint sequence is 0-satisfiable. ◀

Proof of Theorem 6. The max-automaton will accept a constraint sequence iff it is consistent and has no infinitely decreasing one-sided chains and no trespassing one-sided chains of unbounded depth. By Lemma 4, such a sequence is satisfiable.

The max-automaton $A = A_c \wedge A_{\neg\infty} \wedge \big(A_{\neg s} \vee \bigvee_{r\in R}(A^r_m \wedge A^r_{\neg u})\big)$ has five components, and can be described as follows: a constraint sequence is accepted iff it is consistent (A_c), has no infinitely descending chains ($A_{\neg\infty}$), either has no stabilising threads ($A_{\neg s}$) or one of the registers is maximal (A^r_m) and there are no unbounded r-trespassing chains ($A^r_{\neg u}$).

A_c The parity automaton A_c checks consistency, namely that $\forall i\colon \mathit{unprime}(C_{i|R'}) = (C_{i+1})_{|R}$.

$A_{\neg\infty}$ The parity automaton $A_{\neg\infty}$ ensures there are no infinitely decreasing chains. First, we construct the automaton A_∞ that accepts a constraint sequence iff it has such a chain. Intuitively, the automaton guesses such a chain. It starts in the initial state q_0. It loops in q_0 until it nondeterministically decides that now is the starting moment of the chain, in which case it also guesses the first register r_0 of the chain, and it 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. To get $A_{\neg\infty}$, we determinise and complement A_∞.

$A_{\neg s}$ The parity automaton $A_{\neg s}$ accepts a sequence iff it has no stabilising threads, equiv., for every r, the constraints satisfy $(r \neq r')$ infinitely often.

A^r_m Given a register r, the parity automaton A^r_m accepts a sequence iff r is maximal among all stabilising threads. The automaton loops in its initial state until it decides to nondeterministically pick a set $R_s \subseteq R$ with $r \in R_s$ of all stabilising threads and a moment m when all registers in R_s have stabilised, then it verifies that from now on the registers R_s do not change their values while all others do, and that the register r is maximal among R_s. These checks mean that every constraint C read after the moment m contains $r_s = r'_s$ and $r \geq r_s$, for every $r_s \in R_s$; and for every $r_o \notin R_s$, we read C with $r_o \neq r'_o$ infinitely often. A^r_m is parity and can be determinised. It is not hard to show that the result is exponential in $|R|$.

$A^r_{\neg u}$ Given a register r_m, the max-automaton $A^{r_m}_{\neg u}$ ensures the following: if r_m is maximal in a constraint sequence, then there are no increasing or decreasing trespassing one-sided chains of unbounded depth. The automaton $A^{r_m}_{\neg u}$ is a conjunction $B_\searrow \wedge B_\nearrow$ of two automata that check the absence of decreasing and increasing chains. We only describe $B_\searrow$.

The automaton $B_\searrow$ has a set Cn of $|R|$ number of counters. In its state, $B_\searrow$ maintains a partial mapping $cn \colon R \to Cn$. We write $cn(R)$ to denote the counters used by the mapping. Intuitively, in each state of the automaton $B_\searrow$, for each cn-mapped register r, the value of the counter $cn(r)$ reflects the depth of the deepest trespassing decreasing one-sided chain that ends in r in the current moment of the automaton run. We maintain this property of cn during the transition of $B_\searrow$ on reading a constraint C, using operations of max-automata on counters and register-order information from C. On reading C, the automaton does the following:

- Counters releasing. For every r: if $r < r_m$ and $r' > r'_m$, then the automaton performs $reset$ on the counter $cn(r)$ and removes r from the mapping cn.
- Counters allocation. For every r: if $r \geq r_m$ and $r' < r'_m$, then pick a counter $c \in Cn \setminus cn(R)$ and then map $r \mapsto c$ in cn.
- Counters updating. Fix an arbitrary register r such that $r' < r'_m$ holds in C. Let $R^{tre}_{>r'} = \{r_o \mid r_o < r_m \wedge r_o > r'\}$ be the trespassing registers that are larger than the updated r. If $R^{tre}_{>r'}$ is not empty, let $cn(R^{tre}_{>r'})$ be the set of their counters. Let $r_=$ be a register s.t. $r_= = r'$ (may not exist). Then, the automaton does the following operation on the counter $cn(r)$:
  - $reset$ when $R^{tre}_{>r'}$ is empty and $r_=$ does not exist: the condition means that no decreasing trespassing chain can be extended into r';
  - $copy(cn(r_=))$ when $R^{tre}_{>r'}$ is empty and $r_=$ exists: only the chains ending in $r_=$ can be extended into r', and since $r_= = r'$, the deepest chain keeps its depth;
  - $max\big(cn(R^{tre}_{>r'})\big) + 1$ when $R^{tre}_{>r'}$ is not empty and $r_=$ does not exist: the chains from registers in $R^{tre}_{>r'}$ can be extended into r', and since r' is lower than any register in $R^{tre}_{>r'}$, their depths increase. The new value of counter $cn(r)$ reflects the deepest chain.
  - $max\big(max(cn(R^{tre}_{>r'}))+1, cn(r_=)\big)$ when $R^{tre}_{>r'}$ is not empty and $r_=$ exists: some chains from registers in $R^{tre}_{>r'}$ can be decremented into r', and there is also a chain from $r_=$ that can be extended into r' without its depth changed. The updated value of the counter $cn(r)$ reflects the deepest resulting chain.

Thus, $B_\searrow$ moves into the successor state with the updated mapping cn while performing the operations on the counters, as described above. The acceptance condition of $B_\searrow$ requires all counters to be bounded. The number of states of $B_\searrow$ is exponential in $|R|$.

Finally, for the case of 0-satisfiability, the automaton A also needs to satisfy the additional conditions stated in Lemma 5, namely that the constraint sequence starts with C_0 s.t. $C_{0|R} = \{r = s \mid r, s \in R\}$ and that there are no decreasing one-sided chains from moment 0 of depth ≥ 1. These constructions are simple and omitted. ◀

B.6 Proof of Lemma 7

We first prove the following intermediate lemma.

▶ **Lemma 19.** *Suppose a consistent constraint sequence is lasso-shaped and has no trespassing* infinitely *decreasing nor increasing one-sided chains. Then it has no trespassing* unboundedly *decreasing or increasing one-sided chains.*

Proof. If there are no stabilising threads in a given constraint sequence, the proof is trivial, so we assume their presence, and let an r_m-thread be maximal among them. The lasso-shaped constraint sequence has the form $C_0 \dots C_{k-1}(C_k \dots C_{k+l})^\omega$. We focus on the loop $C_k \dots C_{k+l}$. Let $R_m = \{r \mid (r < r_m) \in C_k\}$. Property $(\dagger)$: For every $r \in R_m$, the constraints $C_k \dots C_{k+l}$ cannot have increasing or decreasing chains of depth > 0 starting in r at a moment of reading C_k and ending in r' (primed r) at moment of reading C_{k+l}. I.e., the constraints $C_k \dots C_{k+l}$ cannot require any register $r \in R_m$ to strictly increase (or decrease) along the loop. If this was the case, we would get a trespassing infinitely increasing (or decreasing) one-sided chain, contradicting the lemma premise.

To derive the lemma conclusion, suppose, by contradiction, that there are trespassing unboundedly increasing one-sided chains (the other case is similar). At some point, such chains will start circling the loop $C_k \dots C_{k+l}$ more than $|R|$ times. Consider the registers visited by such a chain at the moments of C_k. The number of registers is $|R|$, but the chain visits C_k more than $|R|$ times, hence some register r is visited twice. I.e., the chain visits r at a moment of reading C_k and visits r again at a moment of reading another C_k later. Since the chain's depth is > 0 and $C_{k|R} = \mathit{unprime}(C_{k+l|R'})$ by consistency of the constraint sequence, we derive a contradiction with the property $(\dagger)$. Hence our assumption is wrong, and the sequence has no trespassing unboundedly increasing chains. The case of decreasing chains is similar. ◀

Proof of Lemma 7. The first item (about satisfiability) follows from Lemma 19 and 4. The second item (about 0-satisfiability) follows from Lemma 19 and 5. ◀

C Proofs of Section 4

C.1 Proof of Lemma 10

Proof. Direction $\Leftarrow$. Let $\lambda : \mathsf{Tst}_R^+ \to \mathsf{Asgn}_R$ be a winning strategy in G. Then we construct λ_f a winning strategy in G_f as follows. For a finite play $\pi = v_0(v_1, \mathsf{tst}_1)(v_2, \mathsf{asgn}_2)\dots(v_n, \mathsf{tst}_n)$, we let $\lambda_f(\pi) = (v_{n+1}, \mathsf{asgn}_{n+1})$ such that $\mathsf{asgn}_{n+1} = \lambda(\mathsf{tst}_1\mathsf{tst}_3 \dots \mathsf{tst}_n)$ and $v_{n+1} = \Delta(v_n, \mathsf{asgn}_n)$. Let $\pi = v_0(v_1, \mathsf{tst}_1)(v_2, \mathsf{asgn}_2)\dots$ be an infinite play compatible with λ_f. Assume that $\overline{a} = \mathsf{tst}_1\mathsf{asgn}_2 \dots$ is a feasible action word. By definition of λ_f, we have that $\overline{a} \in \mathit{Outcome}_G(\lambda)$. By definition of winning strategies in register games, we get that $\pi_{\overline{a}} = v_0v_1v_2\dots$ satisfies the parity condition.

Direction $\Rightarrow$. Let $\lambda_f : \mathit{Plays}_{G_f} \to V_\forall$ be a winning strategy in G_f. We construct a winning strategy $\lambda : \mathsf{Tst}_R^+ \to \mathsf{Asgn}_R$ in G as follows. Let $\rho = \mathsf{tst}_0 \dots \mathsf{tst}_k$ be some sequence of tests. For all $i \in \{0, \dots, k-1\}$, we define $v_0, u_0, v_1, u_1, \dots, v_k \in V$ and $\mathsf{asgn}_1, \dots, \mathsf{asgn}_k$ inductively, such that v_0 is the initial vertex of G and for all $0 \le i \le k$, $u_i = \Delta(v_i, \mathsf{tst}_i)$, $\mathsf{asgn}_{i+1} = \lambda_f(v_0(u_0, \mathsf{tst}_0)(v_1, \mathsf{asgn}_1)\dots(u_i, \mathsf{tst}_i))$, and $v_{i+1} = \Delta(u_i, \mathsf{tst}_i)$. We finally let $\lambda(\mathsf{tst}_0 \dots \mathsf{tst}_k) = \mathsf{asgn}_{k+1}$. Let us show that λ is winning. Let $\overline{a} = \mathsf{tst}_0\mathsf{asgn}_1\mathsf{tst}_1\mathsf{asgn}_2 \dots \in \mathit{Outcome}_G(\lambda)$ and let $\pi_{\overline{a}} = v_0u_0v_1u_1\dots$. Assume that $\overline{a}$ is feasible, we have to show that $\pi_{\overline{a}}$ satisfies the parity condition. By definition of λ, we also get that $v_0(u_0, \mathsf{tst}_0)(v_1, \mathsf{asgn}_1) \dots \in \mathit{Outcome}_{G_f}(\lambda_f)$. Since λ_f is winning, by definition of W_G, we get that $\pi_{\overline{a}}$ satisfies the parity condition. Hence, λ is winning.

The back-and-forth translation of finite-memory strategies can be done similarly, concluding the proof. ◀

C.2 Proof of Lemma 11

Proof. We briefly recall a few definitions from Section 3. A constraint C relates the values of the registers in the current and next moments; it is a maximal consistent set of atoms of the form $t_1 \bowtie t_2$, where $\bowtie \in \{<,=\}$ and each t_1 and t_2 is a register or a primed register (a primed register describes the register in the next moment). A state constraint relates registers at one moment only, so it does not talk about primed registers. Given constraint C, we write $C_{|R}$ to denote the atoms describing the current moment, and $C_{R'}$ — the next moment. We write $\mathit{unprime}(C_{|R'})$ to denote the atoms of $C_{|R'}$ after renaming $r' \mapsto r$ for every $r' \in R'$. Thus, both $C_{|R}$ and $\mathit{unprime}(C_{|R'})$ are state constraints. A constraint sequence is an infinite sequence of constraints; it is 0-satisfiable if there is a sequence of register valuations starting in 0^R that satisfy all constraints. Now we prove the lemma.

Let $R_d = R \uplus \{r_d\}$, where the register r_d will play a role of the last input data. Let Π be the set of all state constraints on R_d; thus each $\pi \in \Pi$ contains atoms of the form $r \bowtie s$ where $r, s \in R_d$ and $\bowtie \in \{<,=\}$.

Given π, tst, asgn, we define the mapping $\mathit{constr}\colon (\pi, \mathsf{tst}, \mathsf{asgn}) \mapsto C$ 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 π (that relates the registers at the beginning of the step).
- Recall that neither tst nor asgn talk about r_d. For readability, we shorten $(t_1 \bowtie t_2) \in C$ to simply $t_1 \bowtie t_2$, $(* \bowtie r) \in \mathsf{tst}$ to $* \bowtie r$, and $a \le b$ 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 $(r = s) \wedge r, s \notin \mathsf{asgn}$ or $r \in \mathsf{asgn} \wedge (* = s)$ or $r, s \in \mathsf{asgn}$;
  - $r' < s'$ iff $(r < s) \wedge r, s \notin \mathsf{asgn}$ or $(* < s) \wedge r \in \mathsf{asgn} \wedge s \notin \mathsf{asgn}$;
  - $r' = r'_d$ iff $(r = *)$ or $r \in \mathsf{asgn}$;
  - $r' \bowtie r'_d$ iff $(r \bowtie *) \wedge r \notin \mathsf{asgn}$, for $\bowtie \in \{<,>\}$;
- So far we 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 $r \notin \mathsf{asgn}$ or $r \in \mathsf{asgn} \wedge (* = r)$;
  - $r \bowtie r'$ iff $r \in \mathsf{asgn} \wedge (r \bowtie *)$, for $\bowtie \in \{<,>\}$;
- Finally, we relate the values of r_d between the moments. There are two cases.
  - The value of r_d crosses another register: $\exists r \in R\colon (r_d < r) \wedge (* \ge r)$. Then $(r'_d > r_d)$. Similarly for the opposite direction: if $\exists r \in R\colon (r_d > r) \wedge (* \le r)$ 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 constr, every action word $\overline{a} = \mathsf{tst}_0\mathsf{asgn}_0\mathsf{tst}_1\mathsf{asgn}_1\ldots$ can be uniquely mapped to the constraint sequence $\mathit{constr}(\overline{a}) = C_0C_1\ldots$ as follows: $C_0 = \mathit{constr}(\pi_0, \mathsf{tst}_0, \mathsf{asgn}_0)$, set $\pi_1 = \mathit{unprime}(C_{0|R'_d})$, then $C_1 = \mathit{constr}(\pi_1, \mathsf{tst}_1, \mathsf{asgn}_1)$, and so on.
We now prove the statement of the lemma, namely that an action word $\overline{a}$ is feasible iff the constraint sequence $\mathit{constr}(\overline{a})$ is 0-satisfiable.
The proof 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 $\mathscr{v}_0\mathscr{d}_0\mathscr{v}_1\mathscr{d}_1\cdots \in (\mathcal{D}^R \cdot \mathcal{D})^\omega$ such that: if some tst is repeated twice and no assignment is done, then the value $\mathscr{d}$ stays the same. This property is needed because of the last item in the definition of constr where we set $r'_d = r_d$. ◀

C.3 Proof of Lemma 12

Proof. We describe a deterministic (parity or max) automaton F accepting all feasible action words. Let V the deterministic (parity or max) automaton accepting all 0-satisfiable constraint sequences (see Theorems 2 or 6). Our automaton F in its state (q_V, π) tracks the state q_V of V and the state constraint π. From (q_V, π), on reading first tst and then asgn, the automaton creates the constraint $C = \mathit{constr}(\pi, \mathsf{tst}, \mathsf{asgn})$, then simulates V on reading C, which gives q'_V, and updates $\pi' = \mathit{unprime}(C_{|R'_d})$; hence F transits into (q'_V, π'). In the beginning all registers are equal, so the initial state of F is (q_0^V, π_0), where q_0^V is initial for V and $\pi_0 = \{r = s \mid r, s \in R_d\}$. The acceptance is defined by the automaton V. Using the properties of constr and of the automaton V, it is easy to see that the automaton F accepts an action word iff it is feasible. The size of F is exponential in R. ◀

D Example and Proofs of Section 5

D.1 Example

In Figure 5, we revisit the typical example of a server granting requests from a set of clients. Each client has a unique priority $p \in \mathbb{N}$, expressing whether s/he should take precedence over others. Note that the number of clients is thus unbounded a priori. The server is equipped with a buffer of size k, and should ensure that (1) there are never more than k pending requests and (2) every request is eventually granted and (3) when a request is granted, it is the one with highest priority. Requests are represented as the set of input signals $\Sigma_I = \{(\mathrm{req}, p) \mid p \in \mathbb{N}\} \cup \{\mathrm{idle}\}$ and grants by the set of output signals $\Sigma_O = \{(g, i) \mid i \in C\} \cup \{\mathrm{idle}\}$ (server grants client i's request). Each client is modelled by his/her unique priority; (req, p) means that the client with priority p requests the ressource and (grt, p) means that his/her request is granted. As input, idle means that no request is conducted at this moment; as output, that no request is granted.

The latter specification is realisable for instance by the transducer which outputs (grt, p) whenever it reads (req, p) and idle whenever it reads idle (Figure 6). Such specification can be enriched as follows: on the first step, the implementation should output idle, modelling the fact that there is an initial request. This is doable since deterministic register automata are closed under intersection. Then, the specification does not admit any implementation anymore: if, initially, some client with low priority inputs some request, it is necessarily buffered (as the implementation has to initially output idle), and it will then starve forever if, afterwards, clients with higher priority repeatedly send requests to the server. This can be mitigated e.g. by allowing the implementation to break the precedence order finitely many times. Then, an implementation would have to used its two registers as buffers, always granting the pending specification with highest precedence.

D.2 Proof of Theorem 16

Proof. Given a register transducer T we construct a register game G_T such that there exists an o-selecting strategy λ such that $L(T \otimes \lambda) \subseteq L(T)$ iff there exists a finite-state winning strategy for Eve in G_T. To decide the latter, by Theorem 1 it suffices to decide the existence of a winning strategy.

The only difference between a register transducer and a game is that register transducers have input and output labels, and can output the content of a register. Assume a slightly different definition of register transducers, with transitions of type $\delta : Q \times \mathsf{Tst}_R \to 2^{\mathsf{Asgn}_R \times Q}$, and the condition that $(\mathsf{asgn}, q_1), (\mathsf{asgn}, q_2) \in \delta(q, \mathsf{tst})$ implies $q_1 = q_2$. Equivalently, we could assume that $|\Sigma_I| = |\Sigma_O| = 1$ and that T always output the same register. We call this type of register transducer a *register transducer is pre-game form*. Then, it is immediate to see that solving the synthesis problem for an RT T in pre-game form reduces to a register game. The only difference is that we have to split transitions of T into two transitions of the

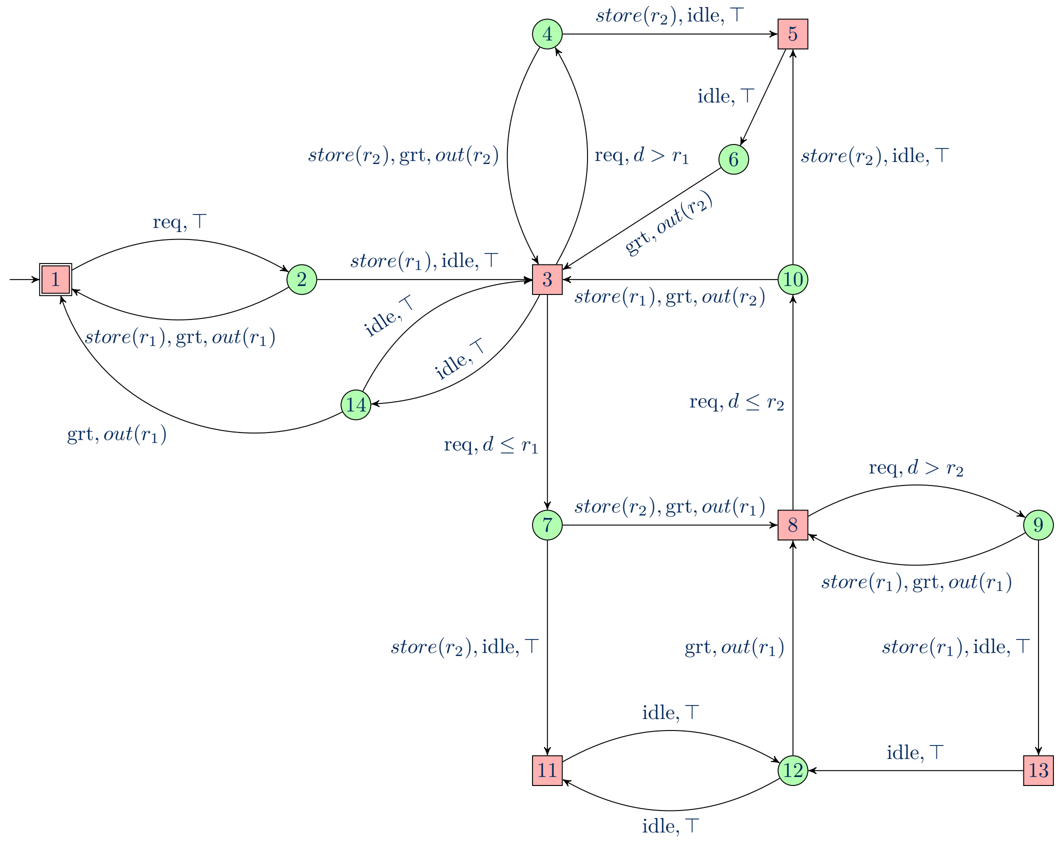


Figure 5 A specification of a server with a buffer of size $k = 2$, ensuring that every request is eventually granted, with precedence to the highest priority. On the output, $out(r)$ means that the transducer outputs the content of r. $\top$ is a macro for two transitions, with respectively $out(r_1)$ and $out(r_2)$.

game: the first where Adam picks the test and the second where Eve picks the assignment. We now show that any register transducer T can be turned into an RT in pre-game form T', such that T is realisable iff T' is realisable. Moreover, winning strategies realising T' can be translated back to winning strategies realising T.

Since the concrete values of the output register do not matter (only the state dynamics of T matters), they can be considered as labels. So, w.l.o.g. we assume that the transitions of T have not output registers. Formally, assume that $T = (\Sigma_I, \Sigma_O, Q, q_0, R, \delta, \alpha)$, where $\delta : Q \times \Sigma_I \times \mathsf{Tst}_R \to 2^{\Sigma_O \times \mathsf{Asgn}_R \times Q}$. Now, we show that we can always assume that $|\Sigma_I| = |\Sigma_O| = 1$ and therefore further assume that δ is of type $\delta : Q \times \mathsf{Tst}_R \to 2^{\mathsf{Asgn}_R \times Q}$, while keeping the property that for all $q, \mathsf{tst}, \mathsf{asgn}$, there exists a unique q' such that $(\mathsf{asgn}, q') \in \delta(q, \mathsf{tst})$. This is done by encoding input and output labels as different data values, which are read by the transducer in an initial phase, and then by replacing occurrences of symbols in Σ_I and Σ_O on the transitions of T as particular tests and assignments respectively. Formally, assume that $\Sigma_I = \{\sigma_1, \dots, \sigma_n\}$ and $\Sigma_O = \{\beta_1, \dots, \beta_m\}$ and they are disjoint. We modify T by asking it to read $n + m$ different data intended to represent the elements of Σ_I and Σ_O respectively. T initially store those $n + m$ data in $n + m$ registers $r_{\sigma_1}, \dots, r_{\sigma_n}, r_{\beta_1}, \dots, r_{\beta_m}$ and during this phase, check that those data are all pairwise different with the test $\mathsf{tst}_{alldiff} = \bigwedge_i r_{\sigma_i} \neq * \wedge \bigwedge_j r_{\beta_j} \neq *$. Then, any transition of the form $t = (q, \sigma_I, \mathsf{tst}, \sigma_O, \mathsf{asgn}, q')$ is replaced by

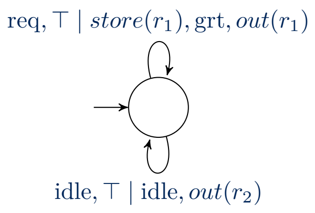

req, $\top$ | $store(r_1)$, grt, $out(r_1)$
idle, $\top$ | idle, $out(r_2)$

Figure 6 A transducer immediately granting each request it receives. On reading idle, the transducer can output anything, here it outputs 0 (the content of r_2).

the two transitions $(q, r_{\sigma_I} = * \wedge \bigwedge_{j \neq i} r_{\sigma_j} \neq *, \mathsf{asgn}' = \{r_{\sigma_O}\}, t)$ and $(t, \mathsf{tst}, \mathsf{asgn}, q')$. The new priority function assigns priority 0 to the newly added states and priority $\alpha(q)$ for the other states $q \in Q$. The new register transducer T' is finally completed by missing transitions (to make its transition function total) to a sink accepting state (with an even priority). This sink state has a loop which for any test, always assign the same assignment (randomly chosen) that we write asgn^*. Note that T' is now in pre-game form.

Claim there exists a winning o-selecting strategy in T iff there exists a winning o-selecting strategy in T'.

Proof From left to right. If there is a winning o-selecting strategy λ in T. Then, it can be turned into the following winning o-selecting strategy λ' in T'. In the initial phase (reading $n+m$ different data), if at some point a test different from $\mathsf{tst}_{alldiff}$ is provided to the strategy λ', then it means that the transducer evolves to the sink accepting state and hence the strategy λ' in this sink state is to always select the unique assignment asgn^*. Otherwise, it means that $n+m$ different data have been provided and after the initial phase the strategy λ' mimics λ. Now, let $t = (q, \sigma_I, \mathsf{tst}, \sigma_O, \mathsf{asgn}, q')$ a transition of T has in the definition of T'. If from $(q, \sigma_I, \mathsf{tst})$ the strategy λ prescribes to select $(\sigma_O, \mathsf{asgn}, q')$, the strategy λ' first from $(q, r_{\sigma_I} = * \wedge \bigwedge_{j \neq i} r_{\sigma_j} \neq *)$ prescribes to select $(\mathsf{asgn}' = \{r_{\sigma_O}\}, t)$ and from (t, tst) it prescribes to select (asgn, q'). The new strategy λ' is winning as it simulates λ which is winning as well. The converse is proved similarly. From a winning strategy λ' we can construct a winning strategy λ which simulates two steps of λ' in one step. *End of Claim Proof.*

Now, let $T' = (Q, q_0, R, \delta, \alpha)$ an RT in pre-game form, where δ has type $\delta : Q \times \mathsf{Tst}_R \to 2^{\mathsf{Asgn}_R \times Q}$ is such that for all $q, \mathsf{tst}, \mathsf{asgn}$, there exists a unique q' such that $(\mathsf{asgn}, q') \in \delta(q, \mathsf{tst})$. We construct the game $G_{T'} = (V_\forall, V_\exists, v_0, \Delta, \alpha')$ as follows:

- $V_\forall = Q$
- $V_\exists = Q \times \mathsf{Tst}_R$
- $v_0 = q_0$
- $\Delta(q, \mathsf{tst}) = (q, \mathsf{tst})$
- $\Delta((q, \mathsf{tst}), \mathsf{asgn})$ is the unique q' such that $(\mathsf{asgn}, q') \in \delta(q, \mathsf{tst})$
- $\alpha'(q) = \alpha'(q, \mathsf{tst}) = \alpha(q)$

Let us briefly sketch why the reduction is correct. Suppose that T' is realisable by some o-selecting strategy λ, i.e. a strategy such that $L(T' \otimes \lambda) \subseteq L(T')$. It is easy to transfer this strategy into a strategy λ' in the register game $G_{T'}$ which naturally simulates λ, because T and T' are almost similar in structure, the only difference being that $G_{T'}$ have split each transition of T into two transitions of Adam and Eve respectively, and tests information are also included in its states. To show that λ' is winning, take an action word $\overline{a} = \mathsf{tst}_0 \mathsf{asgn}_0 \dots$ in the outcome of λ' and suppose it is feasible by some $\mathscr{v}_0 \mathscr{d}_0 \dots$. Since λ' simulates λ, we get that $\mathscr{d}_0 \mathscr{d}_1 \dots \in L(T' \otimes \lambda)$. Since $L(T' \otimes \lambda) \subseteq L(T')$, we get that $\mathscr{d}_0 \mathscr{d}_1 \dots \in L(T')$. This means that the sequence of states $q_0 q_1 \dots$ in T' corresponding to the run $v_0 d_0 \dots$ satisfies the parity condition. By definition of $G_{T'}$, we get that $\pi_{\overline{a}} = q_0(q_0, \mathsf{tst}_0)q_1(q_1, \mathsf{tst}_1)\dots$ and hence by definition of α', $\pi_{\overline{a}}$ satisfies α'. This shows that λ' is winning.

The converse is shown similarly, from a winning strategy λ' in $G_{T'}$ we naturally define a strategy λ in T' which simulates in one-step two steps of λ'. In particular, λ' selects the transitions of T' which corresponds to Eve's choices in $G_{T'}$. It can be shown similarly as before that λ' is also winning.

Moreover, these back-and-forth translations between strategies of T' and strategies of $G_{T'}$ preserves the fact of being finite-memory. Finally, the construction of T' can be done in polynomial time, and therefore the whole procedure runs (construction of T', construction of G_T and solving G_T) in ExpTime. ◀