Transducer Synthesis from Universal Register Automata in $(\mathbb{N}, \leq)$

Ayrat Khalimov
Université libre de Bruxelles, Belgium

Emmanuel Filiot
Université libre de Bruxelles, Belgium

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

Abstract

This paper concerns the problem of reactive synthesis of systems interacting with its environment via letters from an infinite data domain. Register automata and transducers are popular formalisms for specifying and modeling such systems. They extend finite-state automata by introducing registers that are used to store data and to test incoming data against the stored one. Unlike the standard automata, the expressive power of register automata depends on whether they are deterministic, non-deterministic, or universal. Among these, universal register automata suit synthesis best as they can specify request-grant properties and allow for succinct conjunction. Because the synthesis problem from universal register automata is undecidable, researchers studied a decidable variant, called register-bounded synthesis, where additionally a bound on the number of registers in a sought transducer is given. In those synthesis works, however, automata can only compare data for equality, which limits synthesis applications. In this paper, we extend register-bounded synthesis to domains with the natural order $\leq$. To this end, we prove a key Pumping Lemma for a finite abstraction of behaviour of register automata, which allows us to reduce synthesis to synthesis of register-free transducers. As a result, we show that register-bounded synthesis from universal register automata over domain $(\mathbb{N}, \leq)$ is decidable in 2ExpTime as in the equality-only case, giving the order for free.

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

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

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Synthesis aims at automatic construction of a system from its specification. A system is usually modelled as a transducer: in each step, it reads an input from the environment and produces an output. In this way, the transducer, reading an infinite sequence of inputs, produces an infinite sequence of outputs. Specifications are modeled as a language of desirable input-output sequences. Traditionally [15, 2], the inputs and outputs have been modeled as letters from a finite alphabet. This however limits the application of synthesis, so recently researchers started investigating synthesis of systems working on data domains [6, 11, 7, 12, 1, 8].

In the field of automata theory for data languages, a well-studied formalism for specifying and modeling data systems are register automata and register transducers. Register automata extend classic finite-state automata to infinite alphabets $\mathcal{D}$ by introducing a finite number of *registers* [10]. In each step, the automaton reads a data from $\mathcal{D}$, compares it with the values held in its registers, then depending on this comparison it decides to store the data into some of its registers, and finally moves to a successor state. This way it builds a sequence of

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

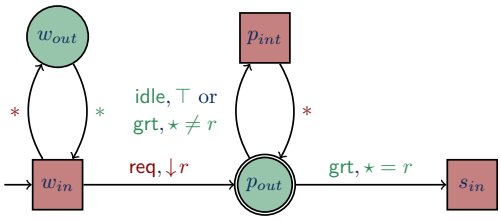

Figure 1 Universal register automaton expressing the specification that any grant must be eventually granted. It reads words over a finite alphabet extended with a data in an infinite countable domain. In this example, data words are assumed to be sequences in $(\{\mathsf{req}, \mathsf{grt}, \mathsf{idle}\} \times \mathbb{N})^\omega$ and the data represent client ids. An incoming pair (req, i) means that client i has requested the resource, while (grt, i) means that it has been granted to client i. The label idle means that no request has been made and in that case the data value is meaningless. States are divided into input states (square) which only read input pairs in $\{\mathsf{req}, \mathsf{idle}\} \times \mathbb{N}$ and are controlled by the environment, and output states (circle) reading pairs in $\{\mathsf{grt}, \mathsf{idle}\} \times \mathbb{N}$, controlled by the system. Transitions are labeled by $*$ or $*$ has a shortcut for any possible input or output pair respectively. Other transitions are labeled with a test on the finite label and a test on the current data (e.g. $\star \neq r$ tests whether the current data is different from that held in r. Additionally, transitions can assign the current data to some registers (e.g. $\downarrow r$ means that the current data is stored in register r). Transitions are universal, and the double circle state is a rejecting state, which has to be visited finitely often (this is easily encoded as a parity condition). There is always an (accepting) run in waiting mode (alternating between states w_{in} and w_{out}). Whenever a request is received, the client id is stored and the automaton universally moves to a pending-request mode, staying in states p_{in} and p_{out} as long as the request is not granted. If it is eventually granted (transition p_{out} to s_{in}), the run dies, and the execution is accepted. Such a specification is not expressible by a non-deterministic register automaton, because any new incoming client id requested the resource must be stored in the registers, to later on check that eventually the request is granted. It is not possible in a single run (even non-deterministically chosen), while universal automata can rely on universal transitions to focus independently on every client request.

configurations (pairs of state and register values) representing its run on reading a word from $\mathcal{D}^\omega$: it is accepted if the visited states satisfy a certain condition, e.g. parity. Transducers are similar except that in each step they also output the content of one register.

Unlike classic finite-state automata, the expressive power of register automata depends on whether they are deterministic, universal, or non-deterministic. Among these, universal register automata (URA) suit synthesis best. First, they can specify request-grant properties: every data labeled with request shall be eventually output labeled with grant. This is the key property in reactive synthesis, it can be expressed by a universal register automaton with a single register, but not by a nondeterministic register automaton. Figure 1 depicts such a universal register automaton. Furthermore, universal register automata are closed, in linear time, under intersection. Therefore they allow to succinctly express conjunctions of (sub)specifications, which is very desirable in synthesis as specifications of reactive systems are usually expressed that way in practice. It is also worth mentioning that universal automata have revealed very useful in the register-free setting to obtain reactive synthesis methods which are feasible in practice [13, 16, 9, 2].

The second factor affecting expressivity of register automata is the allowed comparison operator on data. Originally [10], there were introduced to compare data for equality only, i.e., worked on $(\mathbb{N}, =)$. In this paper, we consider the extension to $(\mathbb{N}, \leq)$.

▶ **Example 1.** We build over the request-grant example of Figure 1 a request-grant example with priorities between clients. A client with id i has a larger priority than any client with id

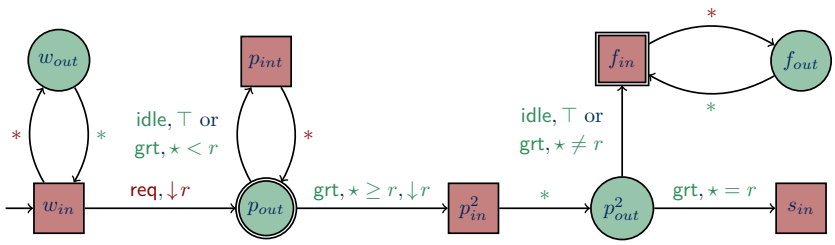

Figure 2 Universal register automaton expressing the specification that any request must be eventually granted to the client of larger or equal id, for two consecutive steps.

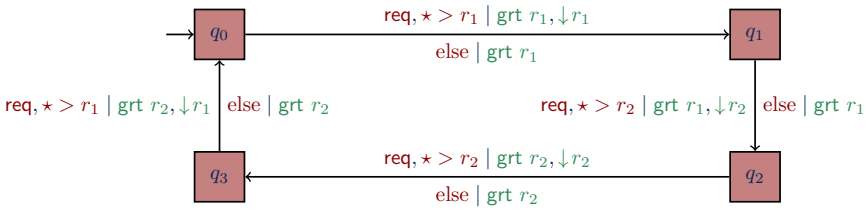

Figure 3 Transducer with two registers r_1, r_2 realizing the specification of Figure 2 (see also Example 1 for a high-level explanation). In red are the tests over the input received by the transducer. The test else is a shortcut for the negation of the other test on the transition. At the right of the vertical bar is the output action performed by the transducer. For example, from state q_0 to q_1, if the input pair is (req, d) and the data d is larger than the data of register r_1, then, the transducer outputs grt with the content of r_1 as data, and then assigns the current data to r_1. Note that there is no acceptance condition, so, such transducers are just an extension of Mealy machines with registers.

$j < i$. Consider the following specification: for all $i \in \mathbb{N}$, whenever client i makes a request, it must eventually be granted to it or a client with larger priority for two consecutive steps. In a first-order like formalism where variables are positions in the word, and can be compared with an order $\leq_p$ while their data can compared by an order $\leq_d$, this would be written:

$$\forall x.(req(x) \to \exists y \geq x.\ data(y) =_d data(y+1)\ \wedge\ data(y) \geq_d data(x)).$$

A URA with one register defining this specification is given on Figure 2. This specification is realizable for instance by a system represented by the register transducer of Figure 3. It has two registers r_1 and r_2 and always perform the sequence of actions: output (grt, r_1) twice, then output (grt, r_2) twice, and so on ad infinitum, alternating between these two instructions. It is correct since r_1 and r_2 satisfy the following invariant: at any point in time, either r_1 or r_2 hold the maximal id of any client who requested the resource. This example could be complexified to make it more realistic and for instance to avoid unsolicited grants of the resource. Examples of register automata and transducers get quickly very large and we rather prefer here to have an minimalistic illustrative example.

Finally, note that if we replace $\geq_d$ by $=_d$ in the first-order formula above, i.e., if we ask that any client making a request gets the resource granted eventually for two consecutive steps, then the specification would be unrealizable by a register transducer in case the clients perform a DDoS attack: at each step, a new client could request the resource, but then a finite number of registers is not enough to remember all client ids who requested the resource.

Synthesis of infinite-alphabet systems is hard. Already for $(\mathbb{N}, =)$, the synthesis of register transducers from universal register automata is undecidable [7]. Decidability is recovered in the deterministic case, both for $(\mathbb{N}, =)$ [7] and $(\mathbb{N}, \leq)$ [8]. As argued above, universal automata

are more desirable in synthesis. To cope with the undecidability issue, researchers studied a decidable variant of synthesis for data domain $(\mathbb{N}, =)$, called register-bounded synthesis, where additionally a bound on the number of registers in a sought transducer is given [11, 7, 12]. Note that the number of states is unconstrained, so, register-bounded synthesis generalizes classical register-free synthesis from (register-free) ω-regular specifications.

Contributions. We extend the results on register-bounded synthesis of [7, 12] from $(\mathbb{N}, =)$ to $(\mathbb{N}, \leq)$. More precisely, we show that given a specification S defined by a universal register automaton over $(\mathbb{N}, \leq)$ and a bound k (in unary), it is decidable (in 2ExpTime) whether there exists a transducer with k registers which realizes S. Moreover, our procedure is effective in the sense that if the specification is realizable, we can output a transducer realizing it (within the same time complexity). If additionally, the bound k and the number of registers in the specification automaton is fixed, the problem is ExpTime-c. These complexity results match the results from [7, 12] (over $(\mathbb{N}, =)$), so we get the order $>$ for free.

Proof techniques. In [7, 12], the key idea is to reduce the problem to a two-player game which finitely abstracts data. Two players alternatively play for an infinite number of rounds. Adam, modelling the environment, picks a label and a test over the k registers, while Eve, modelling the system, replies by picking a label, a register (whose content is meant to be output), and some subset of the k registers (representing the assignment of the selected registers). No data are manipulated in the game, so, it is a finite game arena. Infinite plays in the game induce infinite sequence of (finite) actions by Adam and Eve, called action words, composed of tests, finite labels, and assignments of the k registers. The winning condition is defined in such a way that any finite-memory winning strategy in the game can be converted into a transducer with k registers realizing the specification, and conversely. Since the actions in the game only talk about the k registers, the winning condition require that: (1) sequences of actions over those k registers are *feasible*, in the sense that there exists a data word w satisfying the action sequence (in other words, w is compatible with the action sequence), and (2) that all the runs of the URA defining the specification on data words compatible with the action sequence, are accepting. Checking condition (2) is non-trivial as the URA has its own registers, with no relation whatsoever with the k registers of the sought transducers. In the case of $(\mathbb{N}, =)$, it is known that this winning condition is ω-regular. As already noticed in [8] however, the set of feasible action words in $(\mathbb{N}, \leq)$ is not regular in general. Action words induce infinite sequence of constraints. For example, the action word which alternatively test whether the current data is smaller than r and then assign it to r, induces the infinite sequence of constraints $(r_1 > r_2)(r_2 > r_3)\dots$ where r_i is meant to be register r at time i. Clearly, this (infinite) sequence is not satisfiable in $\mathbb{N}$. Feasibility in $\mathbb{N}$ of an action sequence corresponds to satisfiability in $\mathbb{N}$ of the infinite constraint sequence it induces. In [8], a characterization of constraint sequences satisfiable in $\mathbb{N}$ was established. We rely on this characterization to prove a key pumping lemma. We relax the satisfiability notion (which is non-regular) to a quasi-satisfiability notion (which is regular) and show that any quasi-satisfiable infinite constraint sequence s induces infinitely many satisfiable lasso constraint sequences, in the sense that there are infinitely many decomposition $s = s_1 s_2 s_3$ such that $s_1 s_2^\omega$ is satisfiable in $\mathbb{N}$. The proof of this result is based on a Ramsey argument. Based on this result, we are able to show that quasi-satisfiability is a sufficient notion to solve the two-player game with finite-memory, instead of satisfiability. This allows to reduce our game to a game with an ω-regular condition, which is then decidable.

Structure of the paper. Section 2 introduces some preliminary notions and the main problem. Then we describe the technical tool of constraint sequences in Section 3, and prove the key Pumping Lemma. Finally, Section 4 is dedicated to the proof of the main result, i.e., decidability of register-bounded synthesis from URA in $(\mathbb{N}, \leq)$.

2 Synthesis Problem

Let $\mathbb{N} = \{0, 1, \dots\}$ denote the set of natural numbers including 0. A *data domain* $\mathcal{D}$ is an infinite countable set of elements called *data*, linearly ordered by some order denoted $<$. *We assume the domain order is* non*dense, as in* $\mathbb{N}$. The domain $\mathcal{D}$ has a distinguished element 0: in $\mathbb{N}$ it is the expected zero. *Data words* are infinite sequences $(\sigma_0, d_0)(\sigma_1, d_1)\dots$ of labelled data where $\sigma_i \in \Sigma$ is a label from a finite alphabet Σ and $d_i \in \mathbb{N}$; we sometimes omit the finite-alphabet part of the data part. For a set A, let $A^\infty = A^\omega \cup A^*$ denote a set of finite or infinite sequences of letters of A. We use the following terminology for functions of asymptotic growth: a function is $poly(t)$ if it is $O(t^\kappa)$, $exp(t)$ if it is $O(2^{t^\kappa})$, and $2exp(t)$ if it is $O(2^{2^{t^\kappa}})$, for some constant $\kappa \in \mathbb{N}$. When a function is used with several arguments, the maximal among them shall be used for t.

Register automata. Fix a set of registers R. A *test* is a consistent maximal set of atoms of the form $* \bowtie r$ for $r \in R$ and $\bowtie \in \{=, <, >\}$. We may represent tests as conjunctions of atoms instead of sets. The symbol '$*$' is used as a placeholder for incoming data. For example, for $R = \{r_1, r_2\}$, the expression $r_1 < *$ is not a test because it is not maximal, but $(r_1 < *) \wedge (* < r_2)$ is a test. We denote Tst_R the set of all tests and just Tst if R is clear from the context. A register valuation $\nu \in \mathcal{D}^R$ and data $d \in \mathcal{D}$ *satisfy* a test $\mathsf{tst} \in \mathsf{Tst}$, written $(\nu, d) \models \mathsf{tst}$, if all atoms of tst get satisfied when we replace the placeholder $*$ by d and every register $r \in R$ by $\nu(r)$. An *assignment* is a subset $\mathsf{asgn} \subseteq R$. Given an assignment asgn, a data $d \in \mathcal{D}$, and a 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 *universal register automaton* is a tuple $S = (Q, q_0, R, \delta, \alpha)$ where $Q = Q_\forall \uplus Q_\exists$ is a set of *states* partitioned into Adam's and Eve's states, the state $q_0 \in Q_\forall$ is *initial*, R is a set of *registers*, $\delta = \delta_\forall \uplus \delta_\exists$ is a (total) *transition function* $\delta_p : Q_p \times \mathsf{Tst} \to 2^{\mathsf{Asgn} \times Q_{p'}}$ for $p \in \{\forall, \exists\}$ and the other player p', and $\alpha : Q \to \{1, ..., c\}$ is a *priority function* and c is the *priority index*.

A *configuration* of S is a pair $(q, \nu) \in Q \times \mathcal{D}^R$, describing the state and register content; the *initial configuration* is $(q_0, 0^R)$. A *run* of S on a word $w = d_0 d_1 ... \in \mathcal{D}^\omega$ is a sequence of configurations $\rho = (q_0, \nu_0)(q_1, \nu_1)...$ starting in the initial configuration and such that for every $i \geq 0$: let tst_i be a unique test for which $(\nu_i, d_i) \models \mathsf{tst}_i$, then there exists asgn_i such that $(\mathsf{asgn}_i, q_{i+1}) \in \delta(q_i, \mathsf{tst}_i)$ and $\nu_{i+1} = update(\nu_i, d_i, \mathsf{asgn}_i)$. Notice that a word may induce several runs in S. A run ρ is *accepting* if the maximal priority visited infinitely often is even. A word is *accepted* by S if all its induced runs are accepting. The *language* $L(S)$ of S is the set of all words it accepts. Figure 2 gives an example, where we assume that register automata can read finite-alphabet letters along with data; this can be easily added/modelled in our setting.

Register transducers. A *register transducer* is a tuple $T = (Q, q_0, R, \delta)$, where Q, q_0, and R are as for automata, but $\delta = \delta_\forall \uplus \delta_\exists$ has the components $\delta_\forall : Q_\forall \times \mathsf{Tst} \to \mathsf{Asgn} \times Q_\exists$ and $\delta_\exists : Q_\exists \to R \times Q_A$. A *run* of T on an input word $w = d_0 d_2 d_4 \dots \in \mathcal{D}^\omega$ is a sequence of configurations $\rho = (q_0, \nu_0)(q_1, \nu_1)...$ starting in the initial configuration $(q_0, 0^R)$ and satisfying the following:

- for every even $i \geq 0$: let tst_i be a unique test for which $(\nu_i, d_i) \models \mathsf{tst}_i$ and $\mathsf{asgn}_i = \delta_\forall(q_i, \mathsf{tst}_i)|_{\mathsf{Asgn}}$, then we require $\nu_{i+1} = update(\nu_i, d_i, \mathsf{asgn}_i)$ and $q_{i+1} = \delta_\forall(q_i, \mathsf{tst}_i)|_Q$;
- for every odd $i \geq 1$: $\nu_{i+1} = \nu_i$ and $q_{i+1} = \delta_\exists(q_i)|_Q$.

With every even moment i we can associate a *data input* d_i from the input word w. Similarly, for every odd i, let $r_i = \delta_\exists(q_i)|_R$, then the value $\nu_{i+1}(r_i)$ is called *data output* of T at moment i. The alternating sequence $d_0 d_1 d_2 ...$ of input data and output data is called *input-output* word of T. The language $L(T) \subseteq \mathcal{D}^\omega$ is the set of all input-output words of T. Note that transducer's output is defined for every input word, i.e., their input domain is total. Figure 3 gives an example; again, it assumes that transducers read finite-alphabet letters along data, which is omitted from the definition for simplicity; we merged $Q_\forall$ and $Q_\exists$ states in the figure.

Synthesis problem. A register transducer T *realises* a register automaton A if $L(T) \subseteq L(A)$. The *register-bounded synthesis problem* is:

- input: $k \in \mathbb{N}$ and a URA S;
- output: a k-register transducer realising S, if exists, or else 'unrealisable'.

In Section 4 we will prove that for a URA with n states and c colors, register-bounded synthesis with k registers is solvable in time $exp(exp(r, k), n, c)$.

3 Constraint Sequences

Constraint sequences, consistency and satisfiability

For the rest of this section, fix $\mathcal{D}$. We also fix a finite set of *registers* R. Let $R' = \{r' \mid r \in R\}$ be the set of their *primed* versions. A *constraint* is a maximal consistent set of atoms of the form $t_1 \bowtie t_2$ where $\bowtie \in \{<, >, =\}$ and $t_1, t_2 \in R \cup R'$. Given a constraint C, the writing $C|_R$ denotes the subset of its atoms $r \bowtie s$ for $r, s \in R$, and $C|_{R'}$ — the subset of atoms over primed registers.

A *constraint sequence* is a finite or infinite sequence of constraints $\overline{C} = C_0 C_1 \dots$. It is *consistent* if for every $0 \leq i < |\overline{C}|$, $r, s \in R$, $\bowtie \in \{<, >, =\}$: $r' \bowtie s' \in C_i$ iff $r \bowtie s \in C_{i+1}$. Given a valuation $\nu \in \mathcal{D}^R$, define $\nu' \in \mathcal{D}^{R'}$ to be the valuation that maps $\nu'(r') = \nu(r)$ for every $r \in R$. A valuation $w \in \mathcal{D}^{R \cup R'}$ *satisfies* a constraint C, written $w \models C$, if every atom holds when we replace every $x \in R \cup R'$ by $w(x)$. A constraint sequence $\overline{C}$ is *satisfiable* in $\mathcal{D}$ if there exists a sequence of valuations $\bar{\nu} = \nu_0 \nu_1 ... \in (\mathcal{D}^R)^\infty$ such that $|\overline{C}| = |\bar{\nu}|$ and $\nu_i \cup \nu'_{i+1} \models C_i$ for all $0 \leq i < |\overline{C}|$ and $\nu_0 = 0^R$. Notice that satisfiability implies consistency. Let $\mathcal{C}$ denote the set of constraints over registers R.

Criteria for satisfiability

Let $\overline{C} = C_0 C_1 \dots$ be a consistent constraint sequence over R. A *chain* of $\overline{C}$ is a pair $h = (m, \bar{r})$ where $m \in \mathbb{N}$, $\bar{r} = r_m r_{m+1} \dots \in R^\infty$, $m + |\bar{r}| \leq |\overline{C}|$, and for all $m \leq i < |\bar{r}|$, there exists $\bowtie_i \in \{<, =, >\}$ such that $(r_i \bowtie_i r'_{i+1}) \in C_i$. The *type* of h is the sequence $type(h) = \bowtie_m \bowtie_{m+1} \dots$. The chain h is *infinite* if $|type(h)| = +\infty$ and otherwise it is *finite*, *decreasing* if $type(h) \in \{>, =\}^\infty$, *increasing* if $type(h) \in \{<, =\}^\infty$, and *stable* if $type(h) \in \{=\}^\infty$.

The chains of $\overline{C}$ can be partially ordered as follows: $(m, \bar{r}) \preceq (n, \bar{s})$ if $[m, m + |\bar{r}|] \subseteq [n, n + |\bar{s}|]$ and for all $0 \leq i \leq |\bar{r}|$, the constraint C_{m+i} contains $\bar{r}[i] \lhd \bar{s}[i]$ for some $\lhd \in \{<, =\}$. Note that the relation $\preceq$ is indeed a partial order since $\overline{C}$ is consistent. A *trespassing chain* is a decreasing or increasing chain that is below (for $\preceq$) a stable chain. The *depth* of an increasing (resp. decreasing) chain is the number of atoms $<$ (resp. $>$) in its type and can be infinite.

▶ **Lemma 2** ([8]). *A constraint sequence $\overline{C} = C_0C_1\dots$ is satisfiable iff*
(A) it is consistent, and
(B) it has no decreasing chains of infinite depth, and
(C) there exists $B \in \mathbb{N}$ such that its trespassing chains have a depth at most B, and
(D) $C_0|_R = \{r{=}s \mid r, s \in R\}$, and there is no decreasing chain of depth ≥ 1 from position 0.

Note that when $\overline{C}$ is finite, the conditions *(B)* and *(C)* trivially hold, therefore *(A)* and *(D)* alone characterize satisfiable finite constraint sequences. These two conditions can be easily checked by an automaton, hence the set of satisfiable finite constraint sequences is regular. In contrast, the set of satisfiable infinite constraint sequences is not ω-regular[1], due to condition *(C)*. In [17, Appendix C], this set was shown to be recognizable by a nondeterministic ωB-automaton [3] and in [8]—by a deterministic max-automaton [4]. This represents an obstacle to synthesis as games with such objectives have a high complexity.

Pumping lemma

We prove a key result which allows us to reduce our synthesis problem to an ω-regular game. We replace condition *(C)* (which is not ω-regular) by condition *(C')* which requires the absence of trespassing increasing chains of infinite depth (which is an ω-regular condition). The key result states that any constraint sequence satisfying *(A)+(B)+(C')+(D)* (not necessarily satisfiable) can be decomposed into infinitely many subsequent factors such that the constraint obtained by pumping any factor forever satisfies *(A)+(B)+(C)+(D)*, and hence is satisfiable. We call a constraint sequence *quasi-satisfiable* if it satisfies the conditions *(A)+(B)+(C')+(D)*. It is not hard to show that the set of quasi-satisfiable constraint sequences is ω-regular. We now state the key lemma.

▶ **Lemma 3** (Pumping). *Let $\overline{C} = C_0C_1\dots$ be a quasi-satisfiable infinite constraint sequence. Then, there exists a strictly increasing sequence $(i_j)_{j\in\mathbb{N}} \in \mathbb{N}^\omega$ such that the following constraint sequence is satisfiable for all $j < k$: $C_0C_1\dots C_{i_j-1}\big(C_{i_j}C_{i_j+1}\dots C_{i_k}\big)^\omega$.*

Proof of the Pumping Lemma

The rest of this section is dedicated to the proof of the pumping lemma. We rely on Lemma 2 about characterisation of satisfiable sequences via increasing and decreasing chains, hence we want to track such monotonic chains. That will be the purpose of so-called weak constraints. We will show that weak constraints can be composed: if a weak constraint C_1 tracks monotonic chains from moment i to j, and C_2 – from j to k, then the composition $C_1 \otimes C_2$ tracks monotonic chains from i to k. We will see that the weak constraints with the composition operation $\otimes$ form a finite semigroup. Finally, we will apply a decomposition result from a finite-group theory. Combined together, these give the proof of the pumping lemma.

Decomposition. We start by a decomposition result for infinite words with respect to a finite semigroup. A finite *semigroup* is a pair $(M, \otimes)$ where M is a finite set equipped with an associative operation $\otimes$. An element $e \in M$ is *idempotent* if $e \otimes e = e$. Given an alphabet Σ and a semigroup $(M, \otimes)$, a function $\mu : \Sigma^+ \to M$ is a *morphism* if for all $w_1, w_2 \in \Sigma^+$: $\mu(w_1 \cdot w_2) = \mu(w_1) \otimes \mu(w_2)$.

[1] If it was ω-regular, then we could express the projection of any constraint automaton as an ω-regular language, contradicting [17, Thm.4.3].

▶ **Lemma 4.** *Fix a finite alphabet Σ, a finite semigroup $(M, \otimes)$, a morphism $\mu : \Sigma^+ \to M$. Every $w \in \Sigma^\omega$ can be decomposed into $w = v \cdot w_0 \cdot w_1 \cdot \ldots$ such that $\mu(w_0) = \mu(w_1) = ... = e$ and $e \in M$ is idempotent, where $v \in \Sigma^*$ and every $w_i \in \Sigma^+$.*

Proof. The proof is standard and uses the infinite Ramsey Theorem, so we state it first. Let X be an infinite countable set, $n \in \mathbb{N}$, and $\mathscr{P}(X, n)$ the subsets of X of size n. Let $k \in N_{>0}$ and $c : \mathscr{P}(X, n) \to \{1, \ldots, k\}$ be a (total) function, which can be seen as a colouring of the subsets of X of size n by k colours. The infinite Ramsey Theorem states that there exists an *infinite* subset $Y \subseteq X$ and a colour $e \in \{1, ..., k\}$ such that all subsets of Y of size n have this color e.

To prove the lemma, we take $X = \mathbb{N}$, $n = 2$, and for the colours we use the finite set M. Fix two natural numbers $i < j$, let $P = \{i, j\}$, and let w_P be the factor $w[i{:}j)$ of w from position i (included) to position j (excluded). We colour $P \in \mathscr{P}(\mathbb{N}, 2)$ by $\mu(w_P) \in M$. By the infinite Ramsey Theorem, there exists an infinite set $Y \subseteq \mathbb{N}$ and a colour $e \in M$ such that $\mu(w_P) = e$ for all $P \in \mathscr{P}(Y, 2)$. Notice that e is necessarily idempotent. Indeed, take three numbers $i_1 < i_2 < i_3$ from Y, then: $e \otimes e = \mu\big(w[i_1{:}i_2)\big) \otimes \mu\big(w[i_2{:}i_3)\big) = \mu\big(w[i_1{:}i_2) \cdot w[i_2{:}i_3)\big) = \mu\big(w[i_1{:}i_3)\big) = e$. Finally, let $Y = \{i_0 < i_1 < \ldots\}$, then the sought decomposition is $w = w[0{:}i_0) \cdot w[i_0{:}i_1) \cdot w[i_1{:}i_2) \cdot w[i_2{:}i_3) \cdot \ldots$. ◀

Semigroup of weak constraints. We now define a finite semigroup whose elements finitely abstract sequences of constraints, which will allow us to prove the pumping lemma. This semigroup consists of weak constraints. A *weak constraint* C is either the symbol $\perp$ or a consistent set of atoms of the form $t_1 \bowtie t_2$ for $t_1, t_2 \in R \cup R'$. So, any constraint is a weak constraint, but a weak constraint might not be a constraint since it is does not need to be maximal. Let $\mathscr{W}$ denote the set of weak constraints over registers R.

We now explain how weak constraints can be composed via some multiplication operator $\otimes$. Intuitively, weak constraints track monotonic chains and ignore the others. To capture this intuition, we first introduce a binary operation $\oplus$ on the set $\{<, >, =, ?\}$ as follows: $?$ is absorbing; $=$ is neutral; $< \oplus <$ is $<$ and $> \oplus >$ is $>$; $< \oplus >$ and $> \oplus <$ are $?$. We now define $\otimes$. Fix two weak constraints $C, D \in \mathscr{W}$. If C or D is $\perp$, then $C \otimes D = \perp$. If $C|_{R'} \neq D|_R$, then $C \otimes D = \perp$. Otherwise, given $r, s \in R$, the weak constraint $C \otimes D$ contains $r \bowtie s'$ for $\bowtie \in \{<, >, =\}$ iff there exist $t \in R$ and $\bowtie_1, \bowtie_2 \in \{<, >, =\}$ such that: $(r \bowtie_1 t') \in C$, $(t \bowtie_2 s') \in D$, and $\bowtie_1 \oplus \bowtie_2 = \bowtie$. We also require $(C \otimes D)|_R = C|_R$ and $(C \otimes D)|_{R'} = D|_{R'}$.

▶ **Lemma 5.** *The pair $(\mathscr{W}, \otimes)$ is a finite semigroup.*

Proof. We have to show that $\otimes$ is associative, i.e., that the following equality $(\star)$ holds: $C \otimes (D \otimes E) = (C \otimes D) \otimes E$. Clearly, if one of C, D, E is $\perp$ then the equality holds as $\perp$ is absorbing. Otherwise, we consider several cases. Suppose that $C|_{R'} \neq D|_R$, then $C \otimes D = \perp$ and $(C \otimes D) \otimes E = \perp$. If $D \otimes E = \perp$, then $(\star)$ holds. Otherwise, by definition of $\otimes$, $(D \otimes E)|_R = D|_R$, hence $C|_{R'} \neq (D \otimes E)|_R$ and the result follows. The case $D|_{R'} \neq E|_R$ is shown similarly. Finally, when $C|_{R'} = D|_R$ and $D|_{R'} = E|_R$, the result follows from the associativity of the auxiliary operation $\oplus$ defined earlier. ◀

Weak constraints enable finite abstraction of finite constraint sequences via a function μ we now define. The function $\mu : \mathcal{C}^+ \to \mathscr{W}$ maps a given nonempty finite constraint sequence $\overline{C} = C_0 \ldots C_n$ to a weak constraint $\mu(\overline{C})$ as follows. If $\overline{C}$ is not consistent, then $\mu(\overline{C})$ is $\perp$. Otherwise, we require $\mu(\overline{C})|_R = C_0|_R$ and $\mu(\overline{C})|_{R'} = C_n|_{R'}$. Moreover, for $r, s \in R$ and $\bowtie \in \{<, >, =\}$, $\mu(\overline{C})$ contains $r \bowtie s'$ iff $\overline{C}$ has a chain h from r to s of length $n + 1$ such that: if $\bowtie$ is $<$, then h is increasing and not stable; if $\bowtie$ is $>$, then h is decreasing and not stable; and if $\bowtie$ is $=$, then h is stable.

▶ **Lemma 6.** *The function $\mu : \mathcal{C}^+ \to \mathcal{W}$ is a morphism from nonempty finite constraint sequences to the semigroup $(\mathcal{W}, \otimes)$ of weak constraints.*

Proof. Given two nonempty finite constraint sequences $\overline{S_1}, \overline{S_2}$, we show that $\mu(\overline{S_1}) \otimes \mu(\overline{S_2}) = \mu(\overline{S_1} \cdot \overline{S_2})$. Let $\overline{S_1} = C_0 \dots C_n$ and $\overline{S_2} = C_{n+1} \dots C_m$.

First, observe that $\overline{S_1} \cdot \overline{S_2}$ is not consistent iff either *(i)* $\overline{S_1}$ is not consistent, or *(ii)* $\overline{S_2}$ is not consistent, or *(iii)* they are both consistent but $C_n|_{R'} \neq C_{n+1}|_R$. Consider these three cases separately:

(i) $\mu(\overline{S_1}) \otimes \mu(\overline{S_2}) = \bot \otimes \mu(\overline{S_2}) = \bot = \mu(\overline{S_1} \cdot \overline{S_2})$.

(ii) $\mu(\overline{S_1}) \otimes \mu(\overline{S_2}) = \mu(\overline{S_1}) \otimes \bot = \mu(\overline{S_1} \cdot \overline{S_2})$.

(iii) $\mu(\overline{S_1})|_{R'} = C_n|_{R'} \neq C_{n+1}|_R = \mu(\overline{S_1})|_R$, so by definition of $\otimes$, we have $\mu(\overline{S_1}) \otimes \mu(\overline{S_2}) = \bot$, which is equal to $\mu(\overline{S_1} \cdot \overline{S_2})$ since $\overline{S_1} \cdot \overline{S_2}$ is not consistent.

Now, let us assume that $\overline{S_1} \cdot \overline{S_2}$ is consistent. Let $(r < s') \in \mu(\overline{S_1} \cdot \overline{S_2})$. Therefore there exists an increasing chain from r (at position 0) to s (at the last position) in $\overline{S_1} \cdot \overline{S_2}$, which contains at least one $<$. This chain can be decomposed into an increasing chain from r to some t in $\overline{S_1}$ and an increasing chain from t to s in $\overline{S_2}$. At least one of these two chains contain $<$. Therefore by definition of $\otimes$, we get that $(r < s') \in \mu(\overline{S_1}) \otimes \mu(\overline{S_2})$. Other cases are shown similarly. ◀

We are now able to prove the pumping lemma.

Proof of Pumping Lemma 3. Let $\overline{C} = C_0 C_1 \dots$ be an infinite quasi-satisfiable constraint sequence. We apply Lemma 4 to $\overline{C}$, the semigroup $(\mathcal{W}, \otimes)$ of weak constraints, and the morphism μ. Thus, there exists an idempotent weak constraint E and a decomposition $\overline{C} = \overline{S} \cdot \overline{S_0} \cdot \overline{S_1} \cdot \dots$ such that $\mu(\overline{S_i}) = E$ for all i, where every $\overline{S_i} \in \mathcal{C}^+$ and $\overline{S} \in \mathcal{C}^*$ are constraint sequences. Note that $E \neq \bot$ since $\overline{C}$ is consistent (hence every $\overline{S_i}$ is consistent) and by definition of μ. We now show that for all $i \leq j$ the constraint sequence

$$\overline{C}^{(i,j)} = \overline{S} \cdot \overline{S_0} \cdot \dots \cdot \overline{S_{i-1}} \big(\overline{S_i} \dots \overline{S_j}\big)^\omega$$

is satisfiable, which implies the pumping lemma. It suffices to prove, thanks to Lemma 2, that $\overline{C}^{(i,j)}$ satisfies the conditions *(A)+(B)+(C)+(D)*. We show this by contradiction and distinguish four cases:

1. Suppose that $\overline{C}^{(i,j)}$ does not satisfy *(A)*, i.e., it is not consistent. As $\overline{C}$ is quasi-satisfiable, it is consistent. So, the only reason why $\overline{C}^{(i,j)}$ could be inconsistent is that $\overline{S_j}|_{R'} \neq \overline{S_i}|_R$. It is however impossible since $\mu(\overline{S_j}) = \mu(\overline{S_i}) = E$ is idempotent. Let us see why. If $\overline{S_j}|_{R'} \neq \overline{S_i}|_R$, then $\mu(\overline{S_j} \cdot \overline{S_i}) = \bot$. Since μ is a morphism, we get $\mu(\overline{S_j}) \otimes \mu(\overline{S_i}) = \bot$, so $E \otimes E = \bot$ and as E is idempotent, we get $E = \bot$. And therefore, $\mu(\overline{S_i}) = \bot$. The latter implies that $\overline{S_i}$ is inconsistent and so is $\overline{C}$. Contradiction.
2. Suppose that $\overline{C}^{(i,j)}$ satisfies *(A)* but violates *(B)*, meaning that $\overline{C}^{(i,j)}$ contains a decreasing chain of infinite depth. Hence there exists a decreasing chain from some register x to x with at least one $>$ in $(\overline{S_i} \dots \overline{S_j})^k$ for large enough k. Since $\mu\big((\overline{S_i} \dots \overline{S_j})^k\big) = E$, we get that $(x > x') \in E$. Since $\mu(\overline{S_\ell}) = E$ for all ℓ, every $\overline{S_\ell}$ contains a decreasing chain from x to x. But then we can construct a decreasing chain of infinite depth in $\overline{C}$, which contradicts the fact that $\overline{C}$ satisfies condition *(B)*.
3. Suppose that $\overline{C}^{(i,j)}$ satisfies *(A)* and *(B)* but not *(C)*. Violating *(C)* means that for every bound B, there exists a trespassing (increasing or decreasing) chain T_B of depth $\geq B$. Take $B = p + \ell + 1$, where $p = |\overline{S}\,\overline{S_0} \dots \overline{S_{i-1}}|$ and $\ell = |\overline{S_i} \dots \overline{S_j}|$:

$$\underbrace{\overline{S}\,\overline{S_0} \dots \overline{S_{i-1}}}_{\text{length } p}\ \underbrace{(\overline{S_i} \dots \overline{S_j})}_{\text{length } \ell}\ \underbrace{(\overline{S_i} \dots \overline{S_j})}_{\text{length } \ell} \dots$$

Eventually the chain T_B strictly increases its depth while going through a block $(\overline{S}_i \dots \overline{S}_j)$. Let T_B enter that block through a register x and exit through a register y. Let us assume that T_B is increasing (the other case is similar). Since $\mu(\overline{S}_i \dots \overline{S}_j) = E$, we get $(x < y') \in E$. In our argument, we will need x and y to be the same registers. So we increase the bound to $B = p + |R|\ell + 1$. Then, eventually we will see a sequence of blocks $(\overline{S}_i \dots \overline{S}_j)^k$, for some $k < |R|$, such that T_B enters $(\overline{S}_i \dots \overline{S}_j)^k$ through a register x and exits it through the same x and increases its depth by at least 1. Therefore $(x < x') \in E$. Moreover, since T_B is trespassing, it is below some infinite stable chain of $\overline{C}^{(i,j)}$. By similar arguments, we can show that there exists a register z such that $(z = z') \in E$, $(x \lhd z) \in E$ and $(x' \lhd z') \in E$, where $\lhd \in \{<, =\}$. Coming back to $\overline{C}$, recall that $\overline{C} = \overline{S} \cdot \overline{S_0} \cdot \overline{S_1} \cdot \dots$ and $\mu(\overline{S_\ell}) = E$ for all l. Since $(x < x') \in E$, $(x \lhd z) \in E$, and $(z = z') \in E$, we can construct an increasing chain of infinite depth below a stable chain in $\overline{C}$, contradicting the fact that $\overline{C}$ satisfies condition *(C')*.

Finally, if T_B is decreasing, then, using similar arguments, we can show that there exists a decreasing chain of infinite depth in $\overline{C}$, contradicting the fact that $\overline{C}$ satisfies *(B)*.

4. If $\overline{C}^{(i,j)}$ does not satisfy *(D)*, then $\overline{C}^{(i,j)}$ has a decreasing chain of depth ≥ 1 from position 0. It is not hard to see that the moment when this chain uses $>$ happens on the prefix $\overline{S} \cdot \overline{S_0} \cdot \overline{S_1} \cdot \dots \cdot \overline{S_j}$, which is also a prefix of $\overline{C}$. This contradicts the fact that $\overline{C}$ satisfies condition *(D)*.

◀

4 Solving Register-Bounded Synthesis for URA over $(\mathbb{N}, \leq)$

For readability, we first solve "one-sided" synthesis. There, register transducers read letters from a data domain but output letters from a finite alphabet. The one-sided case contains all necessary proof ingredients. We will lift this restriction afterwards.

A *one-sided URA* is a tuple $A = (\Sigma, Q, q_0, R, \delta, \alpha)$ where the new ingredient Σ is a *finite alphabet*, $\delta = \delta_\forall \uplus \delta_\exists$, $\delta_\forall \subseteq Q_\forall \times \mathsf{Tst} \times \mathsf{Asgn} \times Q_\exists$, and $\delta_\exists \subseteq Q_\exists \times \Sigma \times Q_\forall$. So, the difference is that in Eve's states, the automaton reads a letter from a finite alphabet Σ and does not change the register values. The runs are defined similarly to the case of standard (two-sided) automata. The language of A is a subset of $(\mathcal{D} \cdot \Sigma)^\omega$. In a *one-sided register transducer* $T = (\Sigma, Q, q_0, R, \delta)$, the transition function component $\delta_\forall : Q_\forall \times \mathsf{Tst} \to \mathsf{Asgn} \times Q_\exists$ stays as before while $\delta_\exists : Q_\exists \to \Sigma \times Q_\forall$ has Σ in place of R; the language of T is a subset of $(\mathcal{D} \cdot \Sigma)^\omega$.

The *one-sided register-bounded synthesis problem* is:

- input: $k \in \mathbb{N}$, one-sided URA S;
- output: a one-sided k-register transducer realizing S, if it exists, or else 'unrealisable'.

We will reduce one-sided register-bounded synthesis to classic finite-alphabet synthesis. There, we will synthesize classic register-free transducers having the syntactic structure of register transducers. In other words, we are going to treat the actions of the register transducers as letters of finite input and output alphabets. For example, if we have a register transducer with k registers and output alphabet Σ, then the corresponding classic transducer has an input alphabet Tst_k and an output alphabet $\mathsf{Asgn}_k \times \Sigma$, where Tst_k and Asgn_k are the tests and assignments over k registers.

A *test-assignment sequence* over registers R is $(\mathsf{tst}_0\mathsf{asgn}_0)(\mathsf{tst}_1\mathsf{asgn}_1)\dots \in (\mathsf{Tst}_R \times \mathsf{Asgn}_R)^\omega$; it is *feasible* if there exist a sequence $\nu_0\nu_1\nu_2\dots$ of register valuations and a sequence $\ell_0\ell_1\dots$ of data such that $\nu_0 = 0^R$ and for all plausible j: $\nu_{j+1} = \mathit{update}(\nu_j, \ell_j, \mathsf{asgn}_j)$ and $(\nu_j, \ell_j) \models \mathsf{tst}_j$. Similarly, a transducer *action word* over k registers is a sequence $\bar{a}_k = \mathsf{tst}_0\ (\mathsf{asgn}_0, \sigma_0)\ \mathsf{tst}_1\ (\mathsf{asgn}_1, \sigma_1)\dots \in (\mathsf{Tst}_k \cdot (\mathsf{Asgn}_k \times \Sigma))^\omega$. It is *compatible* with a data

word $w = \mathscr{d}_0\sigma_0\mathscr{d}_1\sigma_1 \ldots \in (\mathscr{D}\Sigma)^\omega$ if there is a sequence $\nu_0\nu_1\nu_2\ldots$ starting in $\nu_0 = 0^R$ such that for all i: $\nu_{i+1} = \mathit{update}(\nu_i, \mathscr{d}_i, \mathsf{asgn}_i)$ and $(\nu_i, \mathscr{d}_i) \models \mathsf{tst}_i$. Given a one-sided URA S, define the language $W_{S,k} \subseteq (\mathsf{Tst}_k \cdot (\mathsf{Asgn}_k \times \Sigma))^\omega$ as

$$W_{S,k} = \{\bar{a}_k \mid \forall w\colon w \text{ is compatible with } \bar{a}_k \Rightarrow w \models S\}.$$

▶ **Lemma 7** (Transfer [7]). *Fix a URA S. These two are equivalent:*

- *S is realisable by a k-register transducer,*
- *$W_{S,k}$ is realisable by a register-free finite-state transducer.*

Intuitively, this lemma holds because every register transducer T can be viewed syntactically as a register-free transducer T', and then one can show that T realises S iff T' realises $W_{S,k}$. We are now departing on the guest to compute $W_{S,k}$.

Fix a one-sided URA S with registers R disjoint from R_k. Let S_{synt} denote the syntactic view of the automaton S: it is a classic register-free automaton with input alphabet $\mathsf{Tst}_S \times \mathsf{Asgn}_S$ and output alphabet Σ. Action words of automata are sequences of the form $\bar{a}_S = (\mathsf{tst}_0^S, \mathsf{asgn}_0^S)\ \sigma_0 \ldots$; the compatibility with data words is defined similarly to the case of transducers. Given a transducer action word $\bar{a}_k = \mathsf{tst}_0^k\ (\mathsf{asgn}_0^k, \sigma_0)\ldots$ and an automaton action word $\bar{a}_S = (\mathsf{tst}_0^S, \mathsf{asgn}_0^S)\ \sigma_0 \ldots$, which agree on Σ letters, define their composition as $\bar{a}_k \otimes \bar{a}_S = (\mathsf{tst}_0^k \uplus \mathsf{tst}_0^S, \mathsf{asgn}_0^k \uplus \mathsf{asgn}_0^S)\ \sigma_0 \ldots$. The composition $\bar{a}_k \otimes \bar{a}_S$ is *feasible* if there exists a data word compatible with both $\bar{a}_k$ and $\bar{a}_S$. Let

$$W_{S,k}^f = \{\bar{a}_k \mid \forall \bar{a}_S\colon \bar{a}_k \otimes \bar{a}_S \text{ is feasible } \Rightarrow \bar{a}_S \in L(S_{synt})\}.$$

▶ **Lemma 8.** $W_{S,k} = W_{S,k}^f$.

Proof. We prove that for every $\bar{a}_k$: $\bar{a}_k \notin W_{S,k} \Leftrightarrow \bar{a}_k \notin W_{S,k}^f$. Consider direction $\Leftarrow$. If $\bar{a}_k \notin W_{S,k}^f$, then $\bar{a}_k \otimes \bar{a}_S$ is feasible for some $\bar{a}_S \notin L(S_{synt})$, hence there is a data word w compatible with both $\bar{a}_k$ and $\bar{a}_S$. Since w is compatible with $\bar{a}_S$, and $\bar{a}_S$ has a rejecting run in S_{synt}, the data word w has a rejecting run in S.

Consider direction $\Rightarrow$. There is a data word w compatible with $\bar{a}_k$, and w has a rejecting run in S. By definition of runs of register automata, there is a sequence $\bar{a}_S = (\mathsf{tst}_0, \mathsf{asgn}_0)\ \sigma_0 \ldots$, induced by running S on w and hence compatible with w, such that $\bar{a}_S \notin L(S_{synt})$. Because the registers of $\bar{a}_S$ and $\bar{a}_k$ are disjoint, their composition $\bar{a}_k \otimes \bar{a}_S$ is feasible, so we are done. ◀

It is possible to show that $W_{S,k}^f$ can be expressed by a nondeterministic ωS-automaton [3] (see Remark 15), however the games with such objectives are not known to be decidable, to the best of our knowledge, and even less is certain about the would-be complexity. Instead, we will find an ω-regular equi-realisable subset of $W_{S,k}^f$, relying on the Pumping Lemma 3.

Let us start with the notion of quasi-feasible action words which is related to quasi-satisfiable constraint sequences. Recall that a constraint C relates the values of the registers between the current moment and the next moment. A *state constraint* relates registers in the current moment only: it contains atoms over non-primed registers, and has no atoms over primed registers. Every test-assignment sequence $(\mathsf{tst}_0\mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1)\ldots$ naturally induces a unique constraint sequence. For instance, for registers $R = \{r, s\}$, a test-assignment sequence starting with $(\{r < *, s < *\}, \{s\})$ (test whether the current data $\mathscr{d}$ is above the values of r and s, store it in s) induces a constraint sequence starting with $\{r = s, r = r', s < s', r' < s'\}$ (the atom $r = s$ is due to all registers being equal initially). This is formalised in the next lemma, which essentially says that given a test-assignment sequence, we can construct a constraint sequence that is satisfiable iff the test-assignment

sequence is feasible. For technical reasons, we need a new register r_d to store the last incoming data, so define $R_d = R \uplus \{r_d\}$.

▶ **Lemma 9** ([8]). *There is a mapping $cstr : \Pi \times \mathsf{Tst} \times \mathsf{Asgn} \to \mathcal{C}$ from state constraints Π over R_d and tests-assignments over R to constraints $\mathcal{C}$ over R_d, such that for all test-assignment sequences $a_0a_1... \in (\mathsf{Tst} \times \mathsf{Asgn})^\omega$, $a_0a_1...$ is feasible iff $C_0C_1...$ is satisfiable, where $\forall i \geq 0$ we have $C_i = cstr(\pi_i, a_i)$ and $\pi_{i+1} = \{r \bowtie s \mid r' \bowtie s' \in C_i\}$, and $\pi_0 = \{r = s \mid r, s \in R_d\}$.*

We overload the notation and use $cstr(\bar{a}_k)$ and $cstr(\bar{a}_S)$ and $cstr(\bar{a}_k \otimes \bar{a}_S)$ to denote the constraint sequences of the corresponding test-assignment sequences. The next claim follows from the definition of feasibility of composed action words and the previous lemma.

▶ **Corollary 10.** *For every $\bar{a}_k$ and $\bar{a}_S$ agreeing on their Σ letters[2]: $\bar{a}_k \otimes \bar{a}_S$ is feasible iff $cstr(\bar{a}_k \otimes \bar{a}_S)$ is satisfiable.*

Given $\bar{a}_k$ and $\bar{a}_S$ agreeing on their Σ letters, we say that a composition $\bar{a}_k \otimes \bar{a}_S$ is *quasi-feasible* if $cstr(\bar{a}_k \otimes \bar{a}_S) = C_0C_1 \ldots$ is quasi-satisfiable. Recall that quasi-satisfiable constraint sequences satisfy the conditions *(A)+(B)+(C')+(D)* from Section 3, and these conditions do not guarantee the satisfiability in general. Let

$$W^{qf}_{S,k} = \{\bar{a}_k \mid \forall \bar{a}_S \colon \bar{a}_k \otimes \bar{a}_S \text{ is quasi-feasible } \Rightarrow \bar{a}_S \in L(S_{synt})\}.$$

▶ **Lemma 11.** *$W^f_{S,k}$ is realisable by a finite-state transducer iff $W^{qf}_{S,k}$ is realisable by a finite-state transducer.*

Proof. If a transducer realises $W^{qf}_{S,k}$, then it also realises $W^f_{S,k}$, because $W^{qf}_{S,k} \subseteq W^f_{S,k}$, as feasibility implies quasi-feasibility.

We now prove the other direction. Let T be a finite-state transducer realising $W^f_{S,k}$. We show that T realises $W^{qf}_{S,k}$. Suppose it does not. Then there exists a word $\bar{a}_k$ of T and a word $\bar{a}_S$ whose composition $\bar{a}_k \otimes \bar{a}_S$ is quasi-feasible yet $\bar{a}_S \notin L(S_{synt})$, meaning that $\bar{a}_S$ induces a rejecting run $\bar{v} = v_0u_0v_1u_1 \ldots \in (Q^S_\forall \cdot Q^S_\exists)^\omega$. Let $\bar{a} = (\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1) \ldots$ be the test-assignment sequence of $\bar{a}_k \otimes \bar{a}_S$; it is quasi-feasible. Let $\overline{C} = cstr(\bar{a}) = C_0C_1 \ldots$ be the quasi-satisfiable constraint sequence induced by $\bar{a}$. By the Pumping Lemma 3, there exist $i_0 < i_1 < i_2 \ldots$ such that for all $j < \ell$, $\overline{C}^{(j,\ell)} = C_0C_1 \ldots C_{i_j-1}(C_{i_j}C_{i_j+1} \ldots C_{i_\ell})^\omega$ is satisfiable. The sequence of indices $i_0 < i_1 < \ldots$ induces a decomposition of $\bar{a}_k$,$\bar{a}_S$, $\bar{a}$, $\bar{v}$ and accordingly, we denote $\bar{a}^{(j,\ell)}_k$, $\bar{a}^{(j,\ell)}_S$, $\bar{a}^{(j,\ell)}$ and $\bar{v}^{(j,\ell)}$ the sequences obtained by pumping the block from index i_j to i_ℓ. For example,

$$\bar{a}^{(j,\ell)} = (\mathsf{tst}_0, \mathsf{asgn}_0) \ldots (\mathsf{tst}_{i_j-1}, \mathsf{asgn}_{i_j-1})\big((\mathsf{tst}_{i_j}, \mathsf{asgn}_{i_j}) \ldots (\mathsf{tst}_{i_\ell}, \mathsf{asgn}_{i_\ell})\big)^\omega.$$

We have that $cstr(\bar{a}^{(j,\ell)}) = \overline{C}^{(j,\ell)}$. Indeed, since $\overline{C}^{(j,\ell)}$ is satisfiable, by Lemma 2 it is consistent, and therefore, $C_{i_\ell}|_{R'} = C_{i_j}|_R$, and the result follows, because the constraint sequence $C_{i_j} \ldots C_{i_\ell}$ induced by $(\mathsf{tst}_{i_j}, \mathsf{asgn}_{i_j}) \ldots (\mathsf{tst}_{i_\ell}, \mathsf{asgn}_{i_\ell})$ uniquely depends on the initial constraint $C_{i_j}|_R$ of its registers.

Now, by choosing suitable j, ℓ, we will show that $\bar{a}^{(j,\ell)}_k$ is a word of T such that $\bar{a}^{(j,\ell)}_k \otimes \bar{a}^{(j,\ell)}_S$ is feasible and $\bar{a}^{(j,\ell)}_S \notin L(S_{synt})$, contradicting the fact that T realises in $W^f_{S,k}$.

First, as $\bar{v}$ does not satisfy the parity condition, the maximal priority seen infinitely often is odd, let us denote it o. Eventually, there exists m such that after m steps, $\bar{v}$ never visits a priority strictly larger than o. So, we have to choose j, ℓ such that $i_j > m$ and moreover,

[2] I.e., whose projection on Σ are equal.

o is seen at least once in the block $\bar{v}[i_j{:}i_\ell)$. There are infinitely many pairs (j,ℓ) satisfying this condition, for infinitely many different j and infinitely many different ℓ. Among this set of pairs, we also pick (j,ℓ) such that the run $\bar{v}$ is in the same state of the automaton S_{synt}, i.e. $v_{i_j} = v_{i_\ell}$ and such that the state of T is the same (this is where the hypothesis that T is finite-state is used). Again, since there are finitely many states in T and S_{synt}, but infinitely many pairs (j,ℓ), we can pick (j,ℓ) satisfying these properties. As a consequence, $\bar{a}_k^{(j,\ell)}$ is a word of T and $\bar{a}_S^{(j,\ell)}$ induces the rejecting run $\bar{v}^{(j,\ell)}$ in S_{synt}. Moreover, their composition $\bar{a}_k^{(j,\ell)} \otimes \bar{a}_S^{(j,\ell)}$ induces the constraint sequence $\overline{C}^{(j,\ell)}$ which is satisfiable according to the Pumping Lemma 3. Hence, $\bar{a}_k^{(j,\ell)} \otimes \bar{a}_S^{(j,\ell)}$ is feasible by Corollary 10. However, the induced run $\bar{v}^{(j,\ell)}$ of S_{synt} on $\bar{a}_S^{(j,\ell)}$ does not satisfy the parity condition, by the choice of (j,ℓ). This contradicts the starting premise that T realises $W^f_{S,k}$. ◀

▶ **Lemma 12.** *There is a universal parity automaton recognising $W^{qf}_{S,k}$; it has $poly(exp(r,k),n)$ states and $poly(r,k,c)$ priorities, where $r = |R_S|$ and $n = |Q_S|$.*

Proof. First, it is not hard to show that there is a deterministic parity automaton A recognising the set of quasi-feasible test-assignment words over registers $R_S \uplus R_k$ with $exp(r,k)$ many states and $poly(r,k)$ many priorities (see Appendix A).

The automaton for $W^{qf}_{S,k}$ is constructed, roughly, as the union $\neg A \vee S_{synt}$. Note that $\neg A \vee S_{synt} = \neg(A \wedge \neg S_{synt})$, where $\neg S_{synt}$ is a nondeterministic automaton with n states and c priorities (dual of S_{synt}). The conjunction of nondeterministic parity automata produces only a polynomial increase in states and priorities, so we get a nondeterministic parity automaton with $poly(exp(r,k),n)$ states and $poly(r,k,c)$ priorities. We then dualise it, and project into the alphabets of inputs Tst_k and outputs $\mathsf{Asgn}_k \times \Sigma$, which gives a universal parity automaton of desired size. We omitted some technical details, namely, ensuring that the alphabets of the automata are in the correct form. ◀

▶ **Proposition 13.** *For a URA in $(\mathbb{N},\leq)$ with n states and c colors, one-sided register-bounded synthesis with k transducer registers is solvable in time $exp(exp(r,k),n,c)$: it is singly exponential in n and c, and doubly exponential in r and k.*

Proof. Fix k and a URA S with n states, c priorities, and r registers. By Lemmas 7, 8, 11, it is sufficient to synthesise a register-free transducer with input alphabet Tst_k and output alphabet $\mathsf{Asgn}_k \times \Sigma$ that realises $W^{qf}_{S,k}$. By Lemma 12, there is a universal parity automaton A with $poly(exp(r,k),n)$ states and $poly(r,k,c)$ priorities. A universal automaton with N states and c priorities can be determinised into an automaton with $exp(N,c)$ states and $poly(N,c)$ priorities (see e.g. [14]). Hence by determinising A, we get a parity game with $exp(exp(r,k),n,c)$ states and $poly(exp(r,k),n,c)$ priorities. The latter can be solved in polynomial time in the number of its states, as the number of priorities is logarithmic in the number of states (see e.g. [5]), giving the overall time complexity $exp(exp(r,k),n,c)$: it is doubly exponential in r and k and singly exponential in n and c. ◀

We now argue how to lift the one-sidedness restriction. The changes are technical and do not pose challenges. Below we highlight the differences for the two-sided case:

- The transducer action words have the form $\bar{a}_k = \mathsf{tst}_0\,(\mathsf{asgn}_0, r_0)\,\mathsf{tst}_1\,(\mathsf{asgn}_1, r_1)\ldots \in (\mathsf{Tst}_k \cdot (\mathsf{Asgn}_k \times R_k))^\omega$; notice R_k instead of Σ. Also, $W_{S,k}, W^f_{S,k}, W^{qf}_{S,k} \subseteq (\mathsf{Tst}_k \cdot (\mathsf{Asgn}_k \times R_k))^\omega$.
- The automaton action words have the form $\bar{a}_S = (\mathsf{tst}^{\mathsf{i}}_0, \mathsf{asgn}^{\mathsf{i}}_0)(\mathsf{tst}^{\mathsf{o}}_0, \mathsf{asgn}^{\mathsf{o}}_0)\ldots \in (\mathsf{Tst}_S \times \mathsf{Asgn}_S)^\omega$, where the components $(\mathsf{tst}^{\mathsf{i}}_i, \mathsf{asgn}^{\mathsf{i}}_i)$ are responsible for dealing with data coming from the environment and the components $(\mathsf{tst}^{\mathsf{o}}_i, \mathsf{asgn}^{\mathsf{o}}_i)$ — from the transducer.

- The definition of the feasibility of composition $\bar{a}_k \otimes \bar{a}_S$ needs to ensure that data coming from a transducer register r_i at moment i in $\bar{a}_k$ satisfies the corresponding test $\mathsf{tst}_i^{\mathrm{o}}$ at moment i of $\bar{a}_S$. This will allow URA to take the transition marked $(\mathsf{tst}_i^{\mathrm{o}}, \mathsf{asgn}_i^{\mathrm{o}})$ at the moment of reading data coming from transducer register r_i.
- In Corollary 10, we need to replace "agreeing on their Σ letters" by "where every r_i satisfies $\mathsf{tst}_i^{\mathrm{o}}$". The latter check should be performed by the automaton recognising $W_{S,k}^{qf}$ constructed in Lemma 12. This does not increase the asymptotic estimate of its size.

After these modifications, the proof of Proposition 13 applies verbatim to the two-sided case.

▶ **Theorem 14.** *For a URA in $(\mathbb{N}, \leq)$ with n states and c colors, register-bounded synthesis with k transducer registers is solvable in time $exp(exp(r, k), n, c)$: it is singly exponential in n and c, and doubly exponential in r and k.*

When the total number of registers $(r + k)$ is fixed, the problem is solvable in $exp(n, c)$ time, matching the known ExpTime-c complexity for synthesis from universal automata over finite alphabets. A tighter complexity analysis is also possible, similar to [12], which shows that the problem is doubly exponential only in r while being singly exponential in k.

▶ Remark 15 (ωS-abstraction). We briefly describe how to construct a so-called ωS automaton [3] for the abstracted specification $W_{S,k}^{f}$. The construction is similar to the one used in the proof of Lemma 12 and, roughly, is a disjunction $\neg A \vee S_{synt}$, where $\neg A$ is an automaton recognising the set of non-feasible (rather than non-quasi-feasible) test-assignment words. The result in [17, Appendix C], when phrased in our notations, describes such an ωS automaton. They are closed under union, hence we get the sought ωS automaton for $W_{S,k}^{f}$.

5 Conclusion

In this paper, we have shown that register-bounded synthesis from specifications expressed by universal register-automata over $(\mathbb{N}, \leq)$ is decidable, within the same time complexity class as the case of URA over $(\mathbb{N}, =)$. This result completes the picture on synthesis from register automata over $(\mathbb{N}, =)$ and $(\mathbb{N}, \leq)$: (unbounded) synthesis is undecidable for non-deterministic register automata [7], decidable for deterministic register automata over $(\mathbb{N}, =)$ [7] and over $(\mathbb{N}, \leq)$ [8], and register-bounded synthesis is decidable for URA over $(\mathbb{N}, =)$ [11, 7, 12] and $(\mathbb{N}, \leq)$ (this paper), and undecidable for non-deterministic register automata [7].

There are challenging future research directions: first, universal automata, as argued in the introduction, are well suited for synthesis, and have been show in the register-free setting to be amenable to synthesis procedures which are feasible in practice [13, 16, 9, 2]. We plan on investigating extensions of these works to the register-setting. In particular, our synthesis algorithm is based on Safra's determinization (for our reduction to finite alphabets), and it is an interesting question whether Safraless procedures from [13, 16, 9] could be combined with our game reduction to get more practical algorithms. Another challenging research direction is to consider synthesis problems from logical specifications instead of automata. The nice correspondences between automata and logics for word languages over finite alphabets however do not carry over to data words. Nevertheless, we believe the expressive power of URA could be useful to target interesting and practical fragments of temporal-like logics with data.

References

1. Béatrice Bérard, Benedikt Bollig, Mathieu Lehaut, and Nathalie Sznajder. Parameterized synthesis for fragments of first-order logic over data words. In *FOSSACS*, volume 12077 of *Lecture Notes in Computer Science*, pages 97–118. Springer, 2020.
2. Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking*, pages 921–962. Springer, 2018.
3. M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 285–296, 2006.
4. Mikołaj Bojańczyk. Weak MSO with the unbounding quantifier. *Theory of Computing Systems*, 48(3):554–576, 2011.
5. C.S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proc. 49th ACM Symp. on Theory of Computing*, pages 252–263, 2017.
6. R. Ehlers, S. Seshia, and H. Kress-Gazit. Synthesis with identifiers. In *Proc. 15th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 8318 of *Lecture Notes in Computer Science*, pages 415–433. Springer, 2014.
7. L. Exibard, E. Filiot, and P-A. Reynier. Synthesis of data word transducers. In *Proc. 30th Int. Conf. on Concurrency Theory*, 2019.
8. Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church Synthesis on Register Automata over Linearly Ordered Data Domains. In Markus Bläser and Benjamin Monmege, editors, *STACS 2021*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2021/13673, doi:10.4230/LIPIcs.STACS.2021.28.
9. E. Filiot, N. Jin, and J.-F. Raskin. An antichain algorithm for LTL realizability. In *Proc. 21st Int. Conf. on Computer Aided Verification*, volume 5643, pages 263–277, 2009.
10. M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
11. 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.
12. Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 25:1–25:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2019.25.
13. O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.
14. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.
15. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.
16. S. Schewe and B. Finkbeiner. Bounded synthesis. In *5th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2007.
17. 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.

A Automaton for recognising quasi-feasible test-assignment words

▶ **Lemma 16.** *There is a deterministic parity automaton recognising the set of quasi-feasible test-assignment words over registers R; it has $exp(|R|)$ many states and $poly(|R|)$ many priorities.*

Proof. First, we construct an automaton A that, using the mapping $cstr$ from Lemma 9, constructs the current constraint and saves it into its successor state; this automaton has $exp(r)$ states, where $r = |R|$. The construction is straightforward and omitted. Now, recall that a test-assignment word $(\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1)\dots$ is not quasi-feasible iff the induced constraint sequence $C_0C_1\dots$ is not quasi-satisfiable, meaning it (*i*) is not consistent, or (*ii*) contains an infinite descending chain, or (*iii*) contains an infinite increasing trespassing chain, or (*iv*) contains a descending chain of depth 1 from C_0. We will construct a nondeterministic parity automaton B that reads constraints (saved on the states of A) and accepts the constraint sequence iff one of the above conditions holds. The automaton B will have $poly(r)$ many states and $O(1)$ many priorities. We then determinise it (see e.g. [14]), which results in a deterministic automaton with $exp(r)$ states and $poly(r)$ priorities, complement it (no blow up), and product with the automaton A. Overall, this gives the sought automaton.

We are left to describe the automaton B. For condition (*i*), the automaton, by using nondeterminism, guesses a position i such that $C_i|_{R'} \neq C_{i+1}|_R$, which can be checked, again, by guessing some atom which is in $C_i|_{R'}$ but not in $C_{i+1}|_R$, or conversely some atom not in $C_i|_{R'}$ but in $C_{i+1}|_R$. This requires only a polynomial number $O(|R|^2)$ of states and a constant number of priorities. We now explain how to check condition (*iii*), as conditions (*ii*) and (*iv*) can be checked using similar ideas. For condition (*iii*), the automaton needs to guess a position i and a first register of the stable chain, below which there will be an infinite increasing chain, also starting from the position i. Starting from the position i, the automaton guesses a next register t of the stable chain and checks that $(s = t') \in C$ belongs to the currently read constraint C, where s is a current register representing the stable chain. We now explain how the automaton ensures the existence of a sought chain. It successively guesses a sequence of registers $r_i, r_{i+1}, \dots$ and checks that $(r_j \leq r'_{j+1}) \in C_j$ and $r_j \leq s \in C_j$ for all $j \geq i$, and checks that infinitely often we see the strict $<$. This needs only $poly(|R|)$ many states and a constant number of priorities. Thus, the nondeterministic automaton B has in total $poly(|R|)$ many states and $O(1)$ many priorities. ◀