Synthesis for Universal Register Automata in Data Domains

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

Emmanuel Filiot
Université libre de Bruxelles, Belgium

Ayrat Khalimov
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 introduce the notion of ω-regular approximable data domains, and show that register-bounded synthesis from universal register automata over such domains is decidable. Importantly, the data domain $(\mathbb{N}, \leq)$ with natural order is ω-regular approximable. We then study data domain $(\mathbb{N}, \leq)$ deeper and show that on it register-bounded synthesis is decidable in 2ExpTime as in the equality-only case, giving the order for free. Finally, we describe the notion of reducibility between data domains, and reduce synthesis in the domain $\mathbb{N}^k$ with partial order $\leq^k$ to $(\mathbb{N}, \leq)$.

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...

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

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 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 *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 requested data shall be eventually output. This is the key property in reactive synthesis, and in the data setting it can be expressed by a universal register automaton but not by a nondeterministic one. Furthermore, universal register automata are closed, in linear time, under intersection. Therefore they allow for succinct conjunction of (sub)specifications, which is very desirable in synthesis as specifications in practice are usually expressed in that way. Moreover, universal automata have revealed to be very useful in the register-free setting for obtaining reactive synthesis methods feasible in practice [13, 16, 9, 2].

The second factor affecting expressivity of register automata is the allowed comparison operator on data. Originally [10], the automata could compare data for equality only, i.e., worked on $(\mathbb{N}, =)$. In this paper, we consider the extension to $(\mathbb{N}, \leq)$. To illustrate these settings, we now consider two examples: the first one is a classical request-grant specification in the domain $(\mathbb{N}, =)$; the second example is an extension to $(\mathbb{N}, \leq)$.

Example 1 (Arbiter in $(\mathbb{N}, =)$)**.** Consider the specification saying that every grant must be eventually granted, expressed by a universal register automaton in Figure 1. The automaton reads words over a finite alphabet extended with a data from an infinite countable domain like $(\mathbb{N}, =)$. In this example, data words are infinite sequences in $[(\{\mathsf{req}, \mathsf{idle}\} \times \mathbb{N}) \cdot (\{\mathsf{grt}, \mathsf{idle}\} \times \mathbb{N})]^\omega$, alternating between input and output letters, where the data component represents client IDs. A pair (req, i) means that client $i \in \mathbb{N}$ 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; the data value is ignored then. 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 $\{\mathsf{grt}, \mathsf{idle}\} \times \mathbb{N}$ and controlled by the system. The transition function of the register automaton is of the form

$$\begin{aligned} Q_\square \times \{\mathsf{req}, \mathsf{idle}\} \times \mathsf{Tst} &\to 2^{\mathsf{Asgn} \times Q_\bigcirc} \quad \uplus \\ Q_\bigcirc \times \{\mathsf{grt}, \mathsf{idle}\} \times \mathsf{Tst} &\to 2^{\mathsf{Asgn} \times Q_\square}. \end{aligned}$$

Here, Tst is a set of conjunctions of atoms of the form $\star = r$ for $r \in R$. For instance, the

test $\star = r$ in state p_{out} holds if the incoming data equals the current value of register r. In the figure, the writing $\top$ means "any test" or "any pair of label-test"; the writing $\star \neq r$ denotes any test that does not contain the atom $\star = r$. The set of assignments is $\mathsf{Asgn} = 2^R$ (where to store the incoming data?), and the assignments are written using notation $\downarrow r$: this means that the incoming data shall be put into register r. The double-circle state is a rejecting state, which can be visited only finitely often (which can be encoded as a parity condition). Thus, if a run loops in wait states w_{in} and w_{out}, it is accepting. Transitions are universal, hence some run is always in one of w_{in} or w_{out}. Whenever a request is received, a copy of the automaton moves to a pending-request mode while storing the client ID into r; it stays 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 run is accepted.

Notice that such a specification is not expressible by a non-deterministic register automaton, because every client ID requesting the resource has to be stored in a register, in order to be able to check that eventually the request is granted. This is not possible in a single run (even non-deterministically chosen), as the automaton has only finitely many registers. In contrast, in universal automata the values of the same register in different runs are independent, so we can devote one independent run to every client ID.

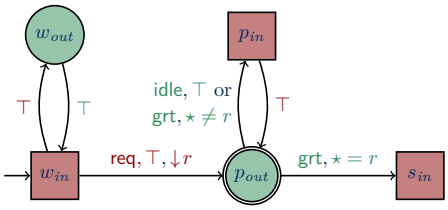

Figure 1 Universal register automaton expressing the specification that every grant is eventually granted (Example 1).

Example 2 (Arbiter in $(\mathbb{N}, \leq)$)**.** We now add priorities between clients: a client with ID i has a larger priority than any client with ID $j < i$. The specification is: for all $i \in \mathbb{N}$, whenever client i requests a resource, it must eventually be granted to it or to a client with a 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 $\leq_{\mathcal{D}}$, this would be written as:

$$\forall x.\ \mathsf{req}(x) \rightarrow \big(\exists y >_p x.\ \mathsf{grt}(y) \wedge \mathsf{grt}(y{+}2) \wedge \mathcal{d}(y) \geq_{\mathcal{D}} \mathcal{d}(x) \wedge \mathcal{d}(y) =_{\mathcal{D}} \mathcal{d}(y{+}2)\big).$$

Here, the writing $\mathcal{d}(y)$ denotes the data value in position y. Note that we compare data at moments y and $y{+}2$ (rather than $y{+}1$) due to alternation of input-output letters in data words. A URA with one register defining this specification is given in Figure 2. In contrast to the previous example, this automaton compares data wrt. $\leq$. This specification is realizable, for instance, by the register transducer in 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.

Finally, note that if we replace $\geq_{\mathcal{D}}$ by $=_{\mathcal{D}}$ in the first-order formula above, i.e., if we ask that every client requesting a resource gets the resource granted eventually for two consecutive steps, then the specification will become unrealizable. Indeed, in this case the clients could perform a DDoS attack, i.e., at each step a new client requests the resource. This scenario

makes the specification unrealisable, because the number of registers in transducers is finite, whereas the number of clients requested but not yet be granted the resource is unbounded.

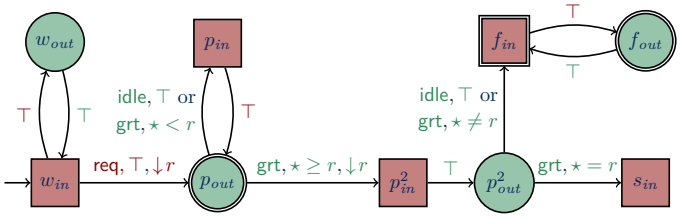

Figure 2 Universal register automaton expressing the specification that every request must eventually be granted to a client with a larger or equal ID, for two consecutive steps.

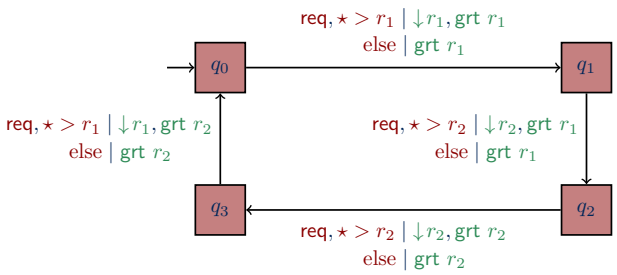

Figure 3 Transducer with two registers r_1, r_2 realizing the specification of Figure 2 from Example 2. In red are the tests over the inputs received by the transducer. The test else is a shortcut for the negation of the other test on the transition. On the right from the vertical bar is the output action performed by the transducer. For example, from state q_0 to q_1, if the input letter is (req, d) and the data d is larger than the data stored in register r_1, then the transducer stores it into r_1, and outputs grt and the content of r_1. Note that there is no acceptance condition, so such transducers are just an extension of Mealy machines by registers.

Reasoning about synthesis over 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 circumvent the undecidability, the works [11, 7, 12] 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. Note that the number of states is still unconstrained, thus register-bounded synthesis generalizes classical register-free synthesis from (register-free) ω-regular specifications.

Contributions. First, we introduce the notion of *ω-regular approximable* data domains and extend known decidability results on synthesis [11, 7, 12] to such domains in general. Importantly, the domain $(\mathbb{N}, \leq)$ falls into such category. Second, for domain $(\mathbb{N}, \leq)$ we prove an upper complexity bound of synthesis. 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 that realizes S. The complexity matches that of [11, 7, 12] (over $(\mathbb{N}, =)$), so we get the order $>$ for free. Finally, we describe the notion of reducibility between data domains, and reduce synthesis in the domain $\mathbb{N}^k$ with partial order $\leq^k$ to $(\mathbb{N}, \leq)$.

2 Synthesis Problem

Let $\mathbb{N} = \{0, 1, \dots\}$ denote the set of natural numbers including 0. In this paper, a *data domain structure* is a tuple $(\mathcal{D}, P, C, c_0)$ consisting of an infinite countable set $\mathcal{D}$ of *data*, a finite set P of interpreted *predicates* (predicate names + arities + their interpretations), a finite set $C \subset \mathcal{D}$ of *constants*, and a distinguished *initializer* constant $c_0 \in C$. For example, $(\mathbb{N}, \{<, =\}, \{0\})$ is the data domain of natural numbers with the usual interpretation of $<$, $=$, and 0; in future, we will denote this domain $(\mathbb{N}, \leq, 0)$. Another familiar example is $(\mathbb{Z}, \{<, =\}, \{0\})$, in future written as $(\mathbb{Z}, \leq, 0)$, which is the data domain of whole numbers with usual $<$, $=$, and 0. *Data words* are infinite sequences $\mathscr{d}_0\mathscr{d}_1 \dots \in \mathcal{D}^\omega$. 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 taken for t.

Action words. Fix a finite set of *registers* R. A register *valuation* is a mapping $\nu : R \to \mathcal{D}$. A test *atom* has the form $p(x_1, ..., x_k)$ where p is a k-arity predicate from P and all $x_i \in R \cup C \cup \{*\}$, where $*$ is a fresh symbol that later will be used as a placeholder for incoming data. A register valuation $\nu : R \to \mathcal{D}$ and data $\mathscr{d} \in \mathcal{D}$ *satisfy* an atom $p(x_1, ..., x_k)$, written $(\nu, \mathscr{d}) \models p(x_1, ..., x_k)$, if the predicate $p(\nu'(x_1), ..., \nu'(x_k))$ holds, where $\nu' = \nu \cup \{c \mapsto c | c \in C\} \cup \{* \mapsto \mathscr{d}\}$. A *test* tst is a set of atoms such that

- (satisfiability) there is $(\nu, \mathscr{d})$ that satisfy all atoms of tst; and
- (maximality) for every atom not in tst, no pair $(\nu, \mathscr{d})$ can simultaneously satisfy tst and a new atom.

Let Tst_R denote the set of all possible tests over registers R in domain $(\mathcal{D}, P, C, c_0)$.

▷*Example.* Consider $(\mathbb{N}, \leq, 0)$. Let $R = \{r\}$. Test atoms are: $r < *$, $* = r$, $r < 0$, $* = 0$, $0 = 0$, $r < r$, etc. We will omit mentioning valid atoms (like $0 = 0$ or $r = r$) and unsatisfiable atoms (like $r < r$). Examples of tests are: $\{r < *, 0 < r, 0 < *\}$, $\{r = *, * = r, 0 = r, r = 0, 0 = *, * = 0\}$. For readability, we will use the formula notation, so we write the first and second tests as $0 < r < *$ and $r = * = 0$ respectively. Examples of non-tests are: $r = * < 0$ (it is not satisfiable), $r < *$ (it is not maximal). ◁

An *assignment* is a mapping $\mathsf{asgn} : R \to R \cup \{*\}$ that commands a register r to take the current value of $\mathsf{asgn}(r)$. Given an assignment asgn, a data $\mathscr{d}$, and a valuation ν, define $update(\nu, \mathscr{d}, \mathsf{asgn})$ to be the valuation that maps every r to $\nu'(\mathsf{asgn}(r))$ where $\nu' = \nu \cup \{* \mapsto \mathscr{d}\}$. Let Asgn_R denote the set of all possible assignments.

A *test-assignment word*, also *action word*, is a sequence $(\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1) \dots \in (\mathsf{Tst}_R \times \mathsf{Asgn}_R)^\omega$. It is *feasible* by a sequence of valuation-data pairs $(\nu_0, \mathscr{d}_0)(\nu_1, \mathscr{d}_1) \dots$ if for all i: $\nu_0 = c_0^R$ and $\nu_{i+1} = update(\nu_i, \mathscr{d}_i, \mathsf{asgn}_i)$, and $(\nu_i, \mathscr{d}_i) \models \mathsf{tst}_i$. Test-assignment word $\bar{a}$ is feasible by $\mathscr{d}_0\mathscr{d}_1 \dots$ if such $\nu_0\nu_1 \dots$ exist, and it is simply feasible if such $\mathscr{d}_i$s and ν_is exist. Let FEAS denote the set of feasible test-assignment words over R in domain $(\mathcal{D}, P, C, c_0)$.

Example.

Register automata. A *register automaton* is a tuple $S = (Q, q_0, R, \delta, \alpha)$, where Q is a finite set of *states* containing the *initial* state q_0, R is a finite set of *registers*, $\delta \subseteq Q \times \mathsf{Tst} \times \mathsf{Asgn} \times Q$ is a *transition relation*, and $\alpha : Q \to \{1, ..., c\}$ is a *priority function* where c is the priority *index*.

A *configuration* of S is a pair $(q, \nu) \in Q \times \mathcal{D}^R$, and (q_0, c_0^R) is *initial*.

A *run* of S on a data word $\mathscr{d}_0\mathscr{d}_1 \dots$ is a sequence of configurations $(q_0, \nu_0)(q_1, \nu_1) \dots$ starting from the initial configuration and such that there exists a test-assignment word

$(\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1)\ldots$ feasible by $(\nu_0, \mathscr{d}_0)(\nu_1, \mathscr{d}_1)\ldots$ and $(q_i, \mathsf{tst}_i, \mathsf{asgn}_i, q_{i+1}) \in \delta$ for all i.

A run $(q_0, \nu_0)(q_1, \nu_1)\ldots$ is *accepting* if the maximal priority appearing infinitely often in $\alpha(q_0)\ \alpha(q_1)\ldots$ is even. A word may induce several runs of S. For *universal* register automata, abbreviated URA, a word is *accepted* if all induced runs are accepting; for *nondeterministic* automata, there should be at least one accepting run. A *language* $L(S)$ is the set of all words accepted by S. Figure 2 gives an example (there, register automata read a data *and* a finite-alphabet letter, which can be added to our setting but we omitted this for readability).

A *finite-alphabet automaton* (without registers) is a tuple $(\Sigma, Q, q_0, \delta, \alpha)$, where Σ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$, and the definition of runs, accepted words, and language is standard. Finite-alphabet automata operate on words from Σ^ω.

Treating a register automaton $S = (Q, q_0, R, \delta, \alpha)$ syntactically gives us a finite-alphabet automaton $S_{synt} = (\Sigma, Q, q_0, \delta, \alpha)$ with $\Sigma = \mathsf{Tst} \times \mathsf{Asgn}$ and the original Q, q_0, δ, α.

Register transducers. A transducer *action word* is a sequence $\mathsf{tst}_0(\mathsf{asgn}_0, r_0)\ldots$ from $\big(\mathsf{Tst}\cdot(\mathsf{Asgn} \times R)\big)^\omega$; it is *feasible* by $(\nu_0, \mathscr{d}_0^{\mathsf{i}}, \mathscr{d}_0^{\mathsf{o}})(\nu_1, \mathscr{d}_1^{\mathsf{i}}, \mathscr{d}_1^{\mathsf{o}})\ldots$ if the test-assignment word $(\mathsf{tst}_0, \mathsf{asgn}_0)\ldots$ is feasible by $(\nu_0, \mathscr{d}_0^{\mathsf{i}})(\nu_1, \mathscr{d}_1^{\mathsf{i}})\ldots$ and $\mathscr{d}_i^{\mathsf{o}} = \nu_{i+1}(r_i)$ for all i. Action word is feasible by $\mathscr{d}_0^{\mathsf{i}}\mathscr{d}_0^{\mathsf{o}}\mathscr{d}_1^{\mathsf{i}}\mathscr{d}_1^{\mathsf{o}}\ldots$ when such ν_is exist, and simply feasible when such $(\nu_i, \mathscr{d}_i^{\mathsf{i}}, \mathscr{d}_i^{\mathsf{o}})$s exist.

A *k-register transducer* is a tuple $T = (Q, q_0, R, \delta)$, where Q, q, R ($|R| = k$) are as in automata but $\delta : Q \times \mathsf{Tst} \ \to \mathsf{Asgn} \times R \times Q$. A *run* of T on an input data word $\mathscr{d}_0^{\mathsf{i}}\mathscr{d}_1^{\mathsf{i}}\ldots$ is a sequence $(q_0, \nu_0)(q_1, \nu_1)\ldots$ starting in the initial configuration such that there exists an action word $\mathsf{tst}_0(\mathsf{asgn}_0, r_0)\ldots$ feasible by $(\nu_0, \mathscr{d}_0^{\mathsf{i}}, \mathscr{d}_0^{\mathsf{o}})(\nu_1, \mathscr{d}_1^{\mathsf{i}}, \mathscr{d}_1^{\mathsf{o}})\ldots$, where $\mathscr{d}_i^{\mathsf{o}} = \nu_{i+1}(r_i)$, and $(\mathsf{asgn}_i, r_i, q_{i+1}) = \delta(q_i, \mathsf{tst}_i)$, for all i. The sequence $\mathscr{d}_0^{\mathsf{o}}\mathscr{d}_1^{\mathsf{o}}\ldots$ is the *output word* of T on reading $\mathscr{d}_0^{\mathsf{i}}\mathscr{d}_1^{\mathsf{i}}\ldots$; it is uniquely defined for every input word. The sequence $\mathscr{d}_0^{\mathsf{i}}\mathscr{d}_0^{\mathsf{o}}\mathscr{d}_1^{\mathsf{i}}\mathscr{d}_1^{\mathsf{o}}\ldots$ is the *input-output word*. The *language* $L(T)$ consists of all input-output words of T. Figure 3 gives an example (there, transducers read a finite-alphabet letter along data, which can also be modelled but omitted for readability).

A *finite-alphabet transducer* is a tuple $(\Sigma_I, \Sigma_O, Q, q_0, \delta)$, where Σ_I and Σ_O are finite input and output alphabets, $\delta : Q \times \Sigma_I \to \Sigma_O \times Q$, and the definition of language is standard. Treating a register transducer T syntactically gives us a finite-alphabet transducer T_{synt} of the same structure with $\Sigma_I = \mathsf{Tst}$ and $\Sigma_O = \mathsf{Asgn} \times R$.

Synthesis problem. Fix a data domain $(\mathscr{D}, P, C, c_0)$. A register transducer T *realises* a register automaton S if $L(T) \subseteq L(S)$. 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'.

3 Recipe for Decidable Synthesis for URA over $(\mathscr{D}, P, C, c_0)$

Given tst and $x, y \in R \cup C \cup \{*\}$, let $\mathsf{tst}[x \leftarrow y]$ denote the set of test atoms derived by substituting x with y in the atoms of tst. Given asgn and $x, y \in R \cup \{*\}$, let $\mathsf{asgn}[x \leftarrow y]$ denote the assignment that maps $r \mapsto \mathsf{asgn}(r)[x \leftarrow y]$, for all $r \in R$.

Given a transducer action word $\bar{a}_k = \mathsf{tst}_0^k\,(\mathsf{asgn}_0^k, r_0^k)\ldots$ and an automaton test-assignment word $\bar{a}_S = (\mathsf{tst}_0^{S\mathsf{i}}, \mathsf{asgn}_0^{S\mathsf{i}})(\mathsf{tst}_0^{S\mathsf{o}}, \mathsf{asgn}_0^{S\mathsf{i}})\ldots$, their *composition* is any test-assignment word $\bar{a}_k \otimes \bar{a}_S = (\mathsf{tst}_0^{\mathsf{i}}, \mathsf{asgn}_0^{\mathsf{i}})(\mathsf{tst}_0^{\mathsf{o}}, \mathsf{asgn}_0^{\mathsf{o}})\ldots$ over $R_k \cup R_S$ satisfying for all j:

- $\mathsf{tst}_j^{\mathsf{i}}$ is any containing the atoms of $\mathsf{tst}_j^{S\mathsf{i}} \cup \mathsf{tst}_j^k$,
- $\mathsf{asgn}_j^{\mathsf{i}} = \mathsf{asgn}_j^{S\mathsf{i}} \cup \mathsf{asgn}_j^k$,
- $\mathsf{tst}_j^{\mathsf{o}}$ is any containing the atoms of $\mathsf{tst}_j^{S\mathsf{o}} \cup \mathsf{tst}_j^{S\mathsf{o}}[* \leftarrow r_j^k]$, and

– $\mathsf{asgn}^{\mathsf{o}}_j = \{r \mapsto r \mid r \in R_k\} \cup \mathsf{asgn}^{S\mathsf{i}}_j[* \leftarrow r^k_j]$.

▷*Example.* Let $R_S = \{s\}$ and $R_k = \{t\}$. Let $\bar{a}_S = (*{>}s, \{\})(*{>}s, * \downarrow s)\ldots$ and $\bar{a}_k = (*{>}t)(*{\downarrow}t, t)\ldots$. Their composition $\bar{a}_k \otimes \bar{a}_S = (\mathsf{tst}^{\mathsf{i}}_0, \mathsf{asgn}^{\mathsf{i}}_0)(\mathsf{tst}^{\mathsf{o}}_0, \mathsf{asgn}^{\mathsf{o}}_0)\ldots$ can have:

– $\mathsf{tst}^{\mathsf{i}}_0$ is either $*{>}s{>}t$ or $*{>}t{>}s$ or $*{>}s{=}t$,
– $\mathsf{asgn}^{\mathsf{i}}_0$ performs $*{\downarrow}t$,
– $\mathsf{tst}^{\mathsf{o}}_0$ must contain $*{>}s$ and $t{>}s$, hence it is either $*{>}t{>}s$ or $t{>}*{>}s$ or $t{=}*{>}s$.
– $\mathsf{asgn}^{\mathsf{o}}_0$ performs $t{\downarrow}s$.

A composition can be $\bar{a}_k \otimes \bar{a}_S = (*{>}s{=}t, * \downarrow t)(t{=}*{>}s, t \downarrow s)\ldots$. Another composition is $\bar{a}_k \otimes \bar{a}_S = (*{>}s{>}t, *{\downarrow}t)\ldots$, but this one is unfeasible in $(\mathbb{N}, \leq, 0)$ as all registers must start in 0, which falsifies the test $*{>}s{>}t$.

◁

Lemma 3. *A transducer and automaton action words $\bar{a}_k$ and $\bar{a}_S$ have a feasible composition, if, and only if, there is a data word making both $\bar{a}_k$ and $\bar{a}_S$ feasible.*

▷**Proof.** Direction $\Rightarrow$. Suppose $\bar{a}_k$ and $\bar{a}_S$ have a composition $\bar{a}_k \otimes \bar{a}_S$ feasible by some $(\nu_0, \mathscr{d}_0)(\nu_1, \mathscr{d}_1)\ldots$. Let the sequence $\mathscr{d}'_0\mathscr{d}'_1\ldots$ be such that for every j: $\mathscr{d}'_{2j} = \mathscr{d}_{2j}$ and $\mathscr{d}'_{2j+1} = \nu_{2j+1}(r^k_j)$; it ignores the original odd values and instead uses the transducer output values. Let ν^S_j be the valuations restricting ν_j to the domain R_S. The sequence $\bar{a}_S$ is feasible by $(\nu^S_0, \mathscr{d}'_0)\ldots$ because: (i) $\bar{a}_k \otimes \bar{a}_S$ subsumes the tests of $\bar{a}_S$, and (ii) the assignments of $\bar{a}_S$ on $\nu^S_0\mathscr{d}'_0\ldots$ and the R_S-assignments of $\bar{a}_k \otimes \bar{a}_S$ on $\nu_0\mathscr{d}_0\ldots$ assign exactly the same values (that is why $\mathscr{d}'_{2j+1}$ is $\nu_{2j+1}(r^k_j)$ instead of $\mathscr{d}_{2j+1}$). Finally, let ν^k_0 be a restriction of ν_0 to the domain R_k, and ν^k_j for all $j \geq 1$ be the R_k-restriction of ν_{2j-1}. Then the sequence $\bar{a}_k$ is feasible by $(\nu^k_0, \mathscr{d}^{\mathsf{i}}_0, \mathscr{d}^{\mathsf{o}}_0)\ldots$, where every $\mathscr{d}^{\mathsf{i}}_j = \mathscr{d}'_{2j}$ and $\mathscr{d}^{\mathsf{o}}_j = \mathscr{d}'_{2j+1}$, because $\bar{a}_k \otimes \bar{a}_S$ subsumes the tests and assignments of $\bar{a}_k$, and $\mathscr{d}^{\mathsf{o}}_j$ equals $\nu^k_{j+1}(r^k_j)$ for all j, as $\mathscr{d}^{\mathsf{o}}_j = \mathscr{d}'_{2j+1} = \nu_{2j+1}(r^k_j) = \nu^k_{j+1}(r^k_j)$. Therefore, both $\bar{a}_S$ and $\bar{a}_k$ are feasible by $\mathscr{d}'_0\mathscr{d}'_1\ldots$.

Direction $\Leftarrow$. Suppose $\bar{a}_k$ and $\bar{a}_S$ are feasible by $\mathscr{d}_0\mathscr{d}_1\ldots$, meaning $\bar{a}_k$ is feasible by $(\nu^k_0, \mathscr{d}^{\mathsf{i}}_0, \mathscr{d}^{\mathsf{o}}_0)\ldots$, where each $\mathscr{d}^{\mathsf{i}}_j = \mathscr{d}_{2j}$ and $\mathscr{d}^{\mathsf{o}}_j = \mathscr{d}_{2j+1}$, while $\bar{a}_S$ is feasible by $(\nu^S_0, \mathscr{d}_0)(\nu^S_1, \mathscr{d}_1)\ldots$. Define the joined registers valuations $\nu_j = \nu^S_j \cup \nu^k_{\lfloor \frac{j+1}{2} \rfloor}$. Let $\bar{a}_k \otimes \bar{a}_S = (\mathsf{tst}^{\mathsf{i}}_0, \mathsf{asgn}^{\mathsf{i}}_0)(\mathsf{tst}^{\mathsf{o}}_0, \mathsf{asgn}^{\mathsf{o}}_0)\ldots$ be the sequence in which the assignments are as in the definition of the composition, $\mathsf{tst}^{\mathsf{i}}_j$ is unique s.t. $(\nu_{2j}, \mathscr{d}_{2j}) \models \mathsf{tst}^{\mathsf{i}}_j$ while $\mathsf{tst}^{\mathsf{o}}_j$ is unique s.t. $(\nu_{2j+1}, \mathscr{d}_{2j+1}) \models \mathsf{tst}^{\mathsf{o}}_j$. Note that assignments $\mathsf{asgn}^{S\mathsf{i}}_j$ and $\mathsf{asgn}^{S\mathsf{i}}_j[* \leftarrow r^k_j]$ command to store exactly the same values since $\mathscr{d}_{2j+1} = \mathscr{d}^{\mathsf{o}}_j = \nu^k_{j+1}(r^k_j) = \nu_{2j+1}(r^k_j)$, i.e., at the moment when those assignments happen the values of incoming data and register r^k_j are the same. Therefore the sequence $\bar{a}_k \otimes \bar{a}_S$ is feasible by $\mathscr{d}_0\mathscr{d}_1\ldots$ with described ν_js. We show that it satisfies the definition of the composition. Since $(\nu^S_{2j}, \mathscr{d}_{2j}) \models \mathsf{tst}^{S\mathsf{i}}_j$, $(\nu^k_j, \mathscr{d}_{2j}) \models \mathsf{tst}^k_j$, $\nu_{2j} = \nu^S_{2j} \cup \nu^k_j$, and $(\nu_{2j}, \mathscr{d}_{2j}) \models \mathsf{tst}^{\mathsf{i}}_j$, $\mathsf{tst}^{\mathsf{i}}_j$ includes the atoms of $\mathsf{tst}^{S\mathsf{i}}_j \cup \mathsf{tst}^k_j$; similarly, $\mathsf{tst}^{\mathsf{o}}_j$ includes the atoms of $\mathsf{tst}^{S\mathsf{o}}_j$. Since $\mathscr{d}^{\mathsf{o}}_j = \nu^k_{j+1}(r^k_j) = \nu_{2j+1}(r^k_j)$, the test $\mathsf{tst}^{\mathsf{o}}_j$ also includes the atoms of $\mathsf{tst}^{S\mathsf{o}}_j[* \leftarrow r^k_j]$. Therefore $\bar{a}_k \otimes \bar{a}_S$ is a feasible composition. ◁

Let $\mathit{Comp}(\bar{a}_k, \bar{a}_S)$ denote the set of all compositions of given $\bar{a}_k$ and $\bar{a}_S$. Define:

$$W^f_{S,k} = \big\{\bar{a}_k \mid \forall \bar{a}_S\colon \mathit{Comp}(\bar{a}_k, \bar{a}_S) \cap \mathsf{FEAS} \neq \varnothing \;\Rightarrow\; \bar{a}_S \in L(S_{synt})\big\}.$$

Lemma 4. *These two are equivalent:*

– *S is realisable by a k-register transducer,*
– *$W^f_{S,k}$ is realisable by a finite-alphabet transducer.*

▷**Proof.** Fix a k-register transducer $T = (Q, q_0, R_k, \delta)$ and a finite-alphabet transducer $T_{synt} = (\mathsf{Tst}_k, \mathsf{Asgn}_k \times R_k, Q, q_0, \delta)$, sharing the same Q, q_0, δ. We show that T does not realise S if, and only if, T_{synt} does not realise $W^f_{S,k}$.

Direction $\Leftarrow$. There is $\bar{a}_k \in L(T_{synt})$ and $\bar{a}_S \notin L(S_{synt})$ having a feasible composition $\bar{a}_k \otimes \bar{a}_S$. Note that $\bar{a}_S \notin L(S_{synt})$ is equivalent to $\bar{a}_S \in L(\overline{S}_{synt})$, where $\overline{S}$ is a nondeterministic register automaton dual to S. By Lemma 3, there is a data word $w = \ell_0\ell_1\ldots$ making $\bar{a}_k$ and $\bar{a}_S$ feasible. Let $q_0q_1\ldots$ be a run of $\overline{S}_{synt}$ on $\bar{a}_S$. Since $\bar{a}_S$ is feasible by w, there is $\nu_0\nu_1\ldots$ such that $\bar{a}_S$ is feasible by $(\nu_0, \ell_0)(\nu_1, \ell_1)\ldots$. The sequence $(q_0, \nu_0)(q_1, \nu_1)\ldots$ satisfies the definition of a run of S on w using $\bar{a}_S$, hence $w \in L(\overline{S})$, i.e. $w \notin L(S)$. Similarly we show that $w \in L(T)$. Therefore, T does not realise S.

Direction $\Rightarrow$. There is a word $w = \ell_0\ell_1\ldots$ such that $w \in L(T)$ and $w \notin L(S)$, hence $w \in L(\overline{S})$. Let $(q_0, \nu_0)(q_1, \nu_1)\ldots$ be a run of $\overline{S}$ on w. By definition of runs, there is a test-assignment word $\bar{a}_S$ feasible by $(\nu_0, \ell_0)(\nu_1, \ell_1)\ldots$, and the sequence $q_0q_1\ldots$ is a run of $\overline{S}_{synt}$ on $\bar{a}_S$. Therefore, $\bar{a}_S \in L(\overline{S}_{synt})$, i.e. $\bar{a}_S \notin L(S_{synt})$. Similarly we show there is $\bar{a}_k \in L(T_{synt})$ feasible by w. By Lemma 3, there is a feasible composition $\bar{a}_k \otimes \bar{a}_S$. As $\bar{a}_S \notin L(S_{synt})$, we have $\bar{a}_k \notin W^f_{S,k}$, thus T_{synt} does not realise $W^f_{S,k}$. ◁

3.1 General Decidability Result

Let *lasso* be the set of lasso-shaped test-assignment sequences over given R. A data domain $\mathcal{D}$ is *good for bounded synthesis* if for every set of registers R, there exists an ω-regular language $\mathsf{QFEAS} \subseteq (\mathsf{Tst}_R \times \mathsf{Asgn}_R)^\omega$ such that

$$\mathsf{QFEAS} \cap \mathit{lasso} \;\subseteq\; \mathsf{FEAS} \;\subseteq\; \mathsf{QFEAS}.$$

Define

$$W^{qf}_{S,k} = \big\{\bar{a}_k \mid \forall \bar{a}_S\colon \mathit{Comp}(\bar{a}_k, \bar{a}_S) \cap \mathsf{QFEAS} \neq \varnothing \;\Rightarrow\; \bar{a}_S \in L(S_{synt})\big\}.$$

Lemma 5. *$W^{qf}_{S,k}$ is ω-regular.*

▷**Proof.** We derive $W^{qf}_{S,k}$ from QFEAS and S_{synt} by applying simple transformations that preserve ω-regularity. For readability, we treat S_{synt} as the language rather than automaton.

- In the complement $\overline{S_{synt}}$, extend the input and output alphabets to $\mathsf{Tst}_R \times \mathsf{Asgn}_R$ and $\mathsf{Tst}_R \times \mathsf{Asgn}_R \times R_k$, where $R = R_k \cup R_S$. In the resulting language, replace asgn in every output letter $(\mathsf{tst}, \mathsf{asgn}, r^k)$ by $\mathsf{asgn}[* \leftarrow r^k]$. Call the result $\overline{S_{synt}}'$.
- Construct the language C over input and output alphabets $\mathsf{Tst}_R \times \mathsf{Asgn}_R$ and $\mathsf{Tst}_R \times \mathsf{Asgn}_R \times R_k$ whose every word $(\mathsf{tst}^{\mathsf{i}}_0, \mathsf{asgn}^{\mathsf{i}}_0)(\mathsf{tst}^{\mathsf{o}}_0, \mathsf{asgn}^{\mathsf{o}}_0, r^k_0)\ldots$ satisfies for all j:
  - $\mathsf{tst}^{\mathsf{o}}_{j\,|R_S}[* \leftarrow r^k] \subset \mathsf{tst}^{\mathsf{o}}_j$, where $\mathsf{tst}^{\mathsf{o}}_{j\,|R_S}$ is the set of atoms of $\mathsf{tst}^{\mathsf{o}}_j$ over $R_S \cup \{*\}$; and
  - $\forall r \in R_k.\ \mathsf{asgn}^{\mathsf{o}}_j(r) = r$ and $\forall r \in R_S.\ \mathsf{asgn}^{\mathsf{o}}_j(r) \in R_S \cup \{r^k_j\}$.

  Therefore, every word of C is a composition $\bar{a}_k \otimes \bar{a}_S$, for some $\bar{a}_k$ and $\bar{a}_S$, extended with output letters R_k.
- Extend the output alphabet of QFEAS from $\mathsf{Tst}_R \times \mathsf{Asgn}_R$ to $\mathsf{Tst}_R \times \mathsf{Asgn}_R \times R_k$. Call the result QFEAS'. Notice that every word in $C \cap \mathsf{QFEAS}'$ is a feasible extended composition for some $\bar{a}_k$ and $\bar{a}_S$.
- Project the intersection $C \cap \mathsf{QFEAS}' \cap \overline{S_{synt}}'$ on the input and output alphabets $\mathsf{Tst}_k \times \mathsf{Asgn}_k$ and R_k; call the result L'.
- In L', shift the component Asgn_k from the input to the output alphabet, resulting in language $L \subseteq \big(\mathsf{Tst}_k \cdot (\mathsf{Asgn}_k \times R_k)\big)^\omega$.

It is not hard to see that L is ω-regular; moreover, $L = \overline{W^{qf}_{S,k}}$. As $\overline{L}$ is ω-regular, $W^{qf}_{S,k}$ is ω-regular as well. ◁

Lemma 6. *$W^f_{S,k}$ is realisable by a finite-state transducer iff $W^{qf}_{S,k}$ is realisable.*

▷**Proof.** Direction $\Leftarrow$ follows from the inclusion $\mathsf{FEAS} \subseteq \mathsf{QFEAS}$ that implies $W^{qf}_{S,k} \subseteq W^f_{S,k}$. Consider direction $\Rightarrow$. We show that if a transducer T does not realise $W^{qf}_{S,k}$, then it does not realise $W^f_{S,k}$. Let S_{synt} be the language of the syntactic view of a given URA S. We will treat T as an ω-regular language over words in $\big(\mathsf{Tst}_k \cdot (\mathsf{Asgn}_k \times R_k)\big)^\omega$. Define the following ω-regular languages.

- Let T'' be the language derived from the language T as follows: the output alphabet changes from $\mathsf{Asgn}_k \times R_k$ to R_k where the component Asgn_k is shifted into the input alphabet. Thus, $T'' \subseteq \big((\mathsf{Tst}_k \times \mathsf{Asgn}_k) \cdot R_k)\big)^\omega$. Now extend the input and output alphabets of T'' to $\mathsf{Tst}_R \times \mathsf{Asgn}_R$ and $\mathsf{Tst}_R \times \mathsf{Asgn}_R \times R_k$, where $R = R_k \cup R_S$. Call the result T'.
- Take the languages $\overline{S_{synt}}', \mathsf{QFEAS}', C \subseteq \big((\mathsf{Tst}_R \times \mathsf{Asgn}_R) \cdot (\mathsf{Tst}_R \times \mathsf{Asgn}_R \times R_k)\big)^\omega$ from the proof of Lemma 5.

The assumption that T does not realise $W^{qf}_{S,k}$ means the language $T' \cap \overline{S_{synt}}' \cap C \cap \mathsf{QFEAS}'$ is nonempty. Since it is ω-regular, it has a lasso word that is a feasible extended composition for some $\bar{a}_k$ and $\bar{a}_S$. Note that $\bar{a}_k \in T$. So we have: $\bar{a}_k \otimes \bar{a}_S \in \mathsf{QFEAS} \cap \mathit{lasso} \subseteq \mathsf{FEAS}$, $\bar{a}_k \in T$, $\bar{a}_S \notin S_{synt}$. Hence $\bar{a}_k \notin W^f_{S,k}$ so T does not realise $W^f_{S,k}$. ◁

Theorem 7. *Register-bounded synthesis for URAs over good $\mathcal{D}$ is decidable.*

▷**Proof.** Follows from Lemmas 5, 6, and the decidability of synthesis wrt. ω-regular languages. ◁

4 In-depth Example for Data Domain $(\mathbb{N}, \leq, 0)$

Criteria for feasibility

We are going to describe a known characterisation [8] of the feasible lasso action words using our notations. Fix a finite set of registers R. An action word $(\mathsf{tst}_0, \mathsf{asgn}_0)\ldots$ is *consistent* if for every i and $r \bowtie s \in \mathsf{tst}_i$: tst_{i+1} contains $r' \bowtie s'$ for every $r' \in \mathsf{asgn}_i^{-1}(r)$ and $s' \in \mathsf{asgn}_i^{-1}(s)$, where $\bowtie \in \{<, =, >\}$, $r, s \in R \cup \{*, 0\}$, and we assume $\mathsf{asgn}_i^{-1}(0) = 0$. In words: two registers at moment $i+1$ are related in the same way as the values they got at moment i.

Let $\bar{a} = (\mathsf{tst}_0, \mathsf{asgn}_0)\ldots$ be a consistent action word. A *chain* of $\bar{a}$ is a pair $(m, \bar{r})$, where $m \in \mathbb{N}$ and $\bar{r} = r_m r_{m+1} \ldots \in R^\infty$, and for every $i \geq m$ there exists $\bowtie_i \in \{<, =, >\}$ such that $r_i \bowtie_i \tilde{r}_{i+1} \in \mathsf{tst}_i$ where $\tilde{r}_{i+1}$ is $\mathsf{asgn}_i(r_{i+1})$. The chain is *infinite* if $|\bar{r}| = \infty$ and otherwise *finite*, *decreasing* if every $\bowtie_i \in \{>, =\}$, *increasing* if every $\bowtie_i \in \{<, =\}$, and *stable* if every $\bowtie_i$ is $=$.

The chains of $\bar{a}$ 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 test tst_{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 $\bar{a}$ 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. $>$) and can be infinite. Figure 4 illustrates the definitions.

Fact 1 ([8]). *An action word $\bar{a} = (\mathsf{tst}_0, \mathsf{asgn}_0)\ldots$ is feasible in $(\mathbb{N}, \leq, 0)$ iff*

(A) it is consistent,
(B) it has no decreasing chains of infinite depth,
(C) there exists $B \in \mathbb{N}$ such that its trespassing chains have a depth at most B, and
(D) tst_0 contains $r = s$ for all $r, s \in R$ and no test contains an atom $* < 0$.

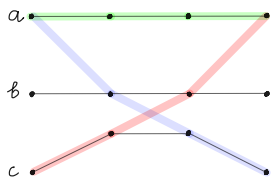

Figure 4 Let $R = \{a, b, c\}$. Illustration of an action word starting $(a > b > * > c, *{\downarrow}c)(a > b > * > c)(a > b > c > *, *{\downarrow}c)$. Assuming that a never changes its value, the green line highlights stable chain $(0, a^\omega)$. The red line highlights increasing trespassing chain $(0, c\,c\,b\,a)$ of depth 3, the blue line – decreasing trespassing chain $(0, a\,b\,c\,c)$ also of depth 3. (We omitted mentioning constants here.)

Due to condition *(C)*, the set of feasible action words is not ω-regular[1]. The result [17, Appendix C] showed that this set is recognizable by a nondeterministic ωB-automaton [3] while [8]—by a deterministic max-automaton [4]. This represents an obstacle to synthesis as games with such objectives have a high complexity. But our interest will be in *lasso-shaped* action words (of the shape uv^ω). To this end, we replace the non-ω-regular condition *(C)* by ω-regular condition *(C')* which requires the absence of trespassing increasing chains of *infinite* depth. Call an action word *quasi-feasible* if it satisfies the conditions *(A)+(B)+(C')+(D)*.

Fact 2 ([8]). *A lasso action word is feasible iff it is quasi-feasible.*

Goodness and synthesis complexity for $(\mathbb{N}, \leq, 0)$

Lemma 8. $(\mathbb{N}, \leq, 0)$ *has a witness of goodness* $\mathsf{QFEAS} \subseteq (\mathsf{Tst}_R \times \mathsf{Asgn}_R)^\omega$ *expressible by a deterministic parity automaton with* $exp(|R|)$ *many states and* $poly(|R|)$ *many priorities.*

▷**Proof.** Define QFEAS to be the set of quasi-feasible action words. It holds that $\mathsf{QFEAS} \cap lasso \subseteq \mathsf{FEAS} \subseteq \mathsf{QFEAS}$: the latter inclusion is obvious, while the first inclusion follows from Fact 2. Hence $(\mathbb{N}, \leq, 0)$ is good. We now show that QFEAS is expressible by a deterministic parity automaton.

First, we construct a nondeterministic parity automaton B that accepts exactly all non-quasi-feasible action words over R. The non-quasi-feasibility means that one of conditions *(A)*–*(C')* is violated. To check the violation of *(A)*, the automaton B, by using nondeterminism, guesses a position i, $r, s \in R \in \{*\}$ and $r', s' \in R$, $\bowtie \in \{<, >, =\}$ falsifying the consistency: $r \bowtie s \in \mathsf{tst}_i$, $r' \in \mathsf{asgn}_i^{-1}(r)$, $s' \in \mathsf{asgn}_i^{-1}(s)$, but $r' \bowtie s' \notin \mathsf{tst}_{i+1}$. This requires only a polynomial number $poly(|R|)$ of states and a constant number of priorities. We now explain how to check the violation of condition *(C')*; the cases of *(B)* and *(D)* are similar. To check the violation of *(C')*, the automaton guesses a position i and a first register of the stable chain, below which there will be an infinite increasing chain, also starting from i. Therefore, starting from i, the automaton guesses a next register s_{i+1} of the stable chain and checks that the atom $(s_i = \tilde{s}_{i+1})$ belongs to the currently read tst_i, where s_i is a current register representing the stable chain and $\tilde{s}_{i+1}$ is $\mathsf{asgn}_i(s_{i+1})$. We now explain how the automaton ensures the existence of a sought infinite chain. It successively guesses a sequence of registers $r_i, r_{i+1}, \ldots$ and checks that $r_j \leq \tilde{r}_{j+1} \in \mathsf{tst}_j$, where $\tilde{r}_{j+1} = \mathsf{asgn}_j(r_{j+1})$, and $r_j \leq s_j \in \mathsf{tst}_j$ for all $j \geq i$, and checks that $<$ appears infinitely often. This requires only $poly(|R|)$ many

[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].

states and a constant number of priorities. Thus, in total, the nondeterministic automaton B has $poly(|R|)$ many states and $O(1)$ many priorities.

We now determinise the automaton B (see e.g. [14]), which results in a deterministic automaton with $exp(|R|)$ states and $poly(|R|)$ priorities, and complement it (no blow up). This gives the sought automaton. ◁

Lemma 9. *For $(\mathbb{N}, \leq, 0)$, 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.** The automaton construction follows the proof structure of Lemma 5. Let $R = R_k \cup R_S$.

Let $\overline{S_{synt}}$ be the nondeterministic automaton dual of S_{synt}. Let $\overline{S_{synt}}'$ be the automaton expressing the language from the proof of Lemma 5. The automaton $\overline{S_{synt}}'$ has n states and c priorities, like S_{synt}.

Let C be the deterministic automaton for the language from the proof of Lemma 5. It has two states and two priorities.

By Lemma 8, there is a deterministic automaton A_{qf} over words in $(\mathsf{Tst}_R \times \mathsf{Asgn}_R)^\omega$ expressing a witness QFEAS of goodness of $(\mathbb{N}, \leq, 0)$; it has $exp(r,k)$ many states and $poly(r,k)$ many priorities. Extend its output alphabet to $\mathsf{Tst}_R \times \mathsf{Asgn}_R \times R_k$; call the result A'_{qf}.

We now conjunct $C \wedge A'_{qf} \wedge \overline{S_{synt}}'$. The conjunction of nondeterministic parity automata results in a polynomial increase in states and priorities, hence $C \wedge A'_{qf} \wedge \overline{S_{synt}}'$ has $poly(exp(r,k), n)$ states and $poly(r,k,c)$ priorities. Note that every word w accepted by $C \wedge A'_{qf} \wedge \overline{S_{synt}}'$ uniquely defines transducer and automaton action words $\bar{a}_k$ and $\bar{a}_S$ having a quasi-feasible composition and $\bar{a}_S \not\models S_{synt}$, implying $\bar{a}_k \notin W^{qf}_{S,k}$.

Project the input alphabet of the nondeterministic automaton $C \wedge A'_{qf} \wedge \overline{S_{synt}}'$ on $\mathsf{Tst}_k \times \mathsf{Asgn}_k$ and output alphabet on R_k; call the result A'. Now, in A' we shift Asgn_k from the input to the output alphabet; call the result A. Note that the language of A is exactly $\overline{W^{qf}_{S,k}}$. As the alphabet shifting may increase the number of states only by factor $exp(k)$, the nondeterministic parity automaton A still has $poly(exp(r,k), n)$ states and $poly(r,k,c)$ priorities, like A'.

Finally, we dualise the nondeterministic parity automaton A, which gives the sought universal parity automaton. ◁

Theorem 10. *For a URA in $(\mathbb{N}, \leq, 0)$ with r registers, 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.*

▷**Proof.** Fix k and a URA S with n states, c priorities, and r registers. By Lemma 6, it is sufficient to synthesise a register-free transducer with input alphabet Tst_k and output alphabet $\mathsf{Asgn}_k \times R_k$ that realises $W^{qf}_{S,k}$. By Lemma 9, 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 $exp(k)$ player actions, $exp(exp(r,k), n, c)$ 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. ◁

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.

5 Reducibility Between Data Domains

A data structure $\mathbb{D}$ *reduces* to a data structure $\mathbb{D}'$ if for every finite set of registers R, there exists a finite set of registers R' and a relation K between R-action words in $\mathbb{D}$ and R'-action words in $\mathbb{D}'$ such that:

1. for every R-action word $\bar{a}$: $\bar{a}$ is feasible in $\mathbb{D}$ iff there exists an R'-action word in $K(\bar{a})$ feasible in $\mathbb{D}'$,
2. if $\bar{a}$ is a lasso, then $K(\bar{a})$ is ω-regular; and
3. if X is ω-regular, then $K^{-1}(X)$ is effectively ω-regular.

Note: conditions (2) and (3) are for instance always satisfied when K is a rational relation[2].

Theorem 11. *If $\mathbb{D}$ reduces to $\mathbb{D}'$ and $\mathbb{D}'$ is good, then $\mathbb{D}$ is good.*

▷**Proof.** Let R be a set of registers that we assume to be fixed in the whole proof and R' be a set of registers satisfying the definition of reducibility. Let FEAS be the set of R-action words feasible in $\mathbb{D}$ and FEAS' — feasible R'-action words in $\mathbb{D}'$.

Our goal is to define an ω-regular set QFEAS s.t. $\mathsf{QFEAS} \cap lasso \subseteq \mathsf{FEAS} \subseteq \mathsf{QFEAS}$. Since $\mathbb{D}'$ is good, there is an ω-regular set QFEAS' s.t. $\mathsf{QFEAS}' \cap lasso \subseteq \mathsf{FEAS}' \subseteq \mathsf{QFEAS}'$. Define $\mathsf{QFEAS} = K^{-1}(\mathsf{QFEAS}')$; it is ω-regular by property 3 of K. We now show that $\mathsf{FEAS} \subseteq \mathsf{QFEAS}$. Before proceeding, notice that property 1 of K is equivalent to $\mathsf{FEAS} = K^{-1}(\mathsf{FEAS}')$ (†). Since $\mathsf{FEAS}' \subseteq \mathsf{QFEAS}'$, we have $K^{-1}(\mathsf{FEAS}') \subseteq K^{-1}(\mathsf{QFEAS}')$, hence $\mathsf{FEAS} \subseteq \mathsf{QFEAS}$.

It remains to show that $\mathsf{QFEAS} \cap lasso \subseteq \mathsf{FEAS}$. The inclusion $\mathsf{QFEAS}' \cap lasso \subseteq \mathsf{FEAS}'$ implies $K^{-1}(\mathsf{QFEAS}' \cap lasso) \subseteq K^{-1}(\mathsf{FEAS}') = \mathsf{FEAS}$ (the latter equality is by †). We prove that $\mathsf{QFEAS} \cap lasso \subseteq K^{-1}(\mathsf{QFEAS}' \cap lasso)$, which entails the desired result. Pick an arbitrary $\bar{a} \in \mathsf{QFEAS} \cap lasso$. By property 2 of K, $K(\bar{a})$ is ω-regular, and since QFEAS' is ω-regular, so is $K(\bar{a}) \cap \mathsf{QFEAS}'$. Since $\bar{a} \in K^{-1}(\mathsf{QFEAS}')$, the intersection $K(\bar{a}) \cap \mathsf{QFEAS}'$ is nonempty. Since $K(\bar{a}) \cap \mathsf{QFEAS}'$ is ω-regular and nonempty, it contains a lasso word $\bar{a}'$. Thus, $\bar{a}' \in K(\bar{a}) \cap \mathsf{QFEAS}' \cap lasso$, hence $\bar{a} \in K^{-1}(\mathsf{QFEAS}' \cap lasso)$. ◁

Corollary 12. *If $\mathbb{D}$ reduces to $\mathbb{D}'$ and $\mathbb{D}'$ is good, then bounded synthesis is decidable for $\mathbb{D}$.*

Example: reducing $(\mathbb{Z}, \leq, 0)$ to $(\mathbb{N}, \leq, 0)$

Lemma 13. *$(\mathbb{Z}, \leq, 0)$ reduces to $(\mathbb{N}, \leq, 0)$.*

▷**Proof.** Fix a set of registers R. We are going to define a binary relation K between R-action words in $(\mathbb{Z}, \leq, 0)$ and R-action words in $(\mathbb{N}, \leq, 0)$ satisfying the definition of reducibility. First, we define a binary relation $\kappa \subseteq \mathsf{Tst}_R^2$ that maps tests for data domains $(\mathbb{Z}, \leq, 0)$ to tests for $(\mathbb{N}, \leq, 0)$: $(\mathsf{tst}, \mathsf{tst}') \in \kappa$ iff for all $r, s \in R \cup \{\star, 0\}$ and $\lhd_1, \lhd_2, \lhd_3, \lhd_4, \lhd_5 \in \{<, =\}$,

– if $(0 \lhd_1 r), (0 \lhd_2 s), (r \lhd_3 s) \in \mathsf{tst}$, then $(0 \lhd_1 r), (0 \lhd_2 s), (r \lhd_3 s) \in \mathsf{tst}'$, and

[2] Rational relations are those recognized by finite non-deterministic transducers. Here K is recognizable by a finite non-deterministic transducer with a single state, as it needs to check that the pairs of input and output symbols it reads is in κ, which is a local property.

– if $(r < 0), (s \lhd_4 0), (r \lhd_5 s) \in \mathsf{tst}$, then $(0 < r), (0 \lhd_4 s), (s \lhd_5 r) \in \mathsf{tst}'$.

Note that κ uniquely defines tst', it is a function. By extending κ to action words, we get K:

$$\big((\mathsf{tst}_0, \mathsf{asgn}_0)\dots,\ (\mathsf{tst}'_0, \mathsf{asgn}'_0)\dots\big) \in K \text{ iff } \forall i.\ \mathsf{asgn}_i = \mathsf{asgn}'_i \wedge (\mathsf{tst}_i, \mathsf{tst}'_i) \in \kappa.$$

Like κ, K is a function. It is easily seen that K is a rational relation, hence it satisfies properties 2 and 3 of the definition of reducibility. Let us prove that K satisfies property 1. Fix a test-assignment word $\bar{a}_{\mathbb{Z}} = (\mathsf{tst}_0, \mathsf{asgn}_0)\dots$.

Suppose that $\bar{a}_{\mathbb{Z}}$ is feasible in $(\mathbb{Z}, \leq, 0)$ by a sequence of valuation-data pairs $\bar{\nu}_{\mathbb{Z}} = (\nu_0, \mathscr{d}_0)(\nu_1, \mathscr{d}_1)\dots$. We construct $\bar{a}_{\mathbb{N}} \in K(\bar{a}_{\mathbb{Z}})$ and a sequence of data-valuations $\bar{\nu}_{\mathbb{N}} = (\nu'_0, \mathscr{d}'_0)\dots$ that makes $\bar{a}_{\mathbb{N}}$ feasible in $(\mathbb{N}, \leq, 0)$. Define $\nu'_i(r) = |\nu_i(r)|$ and $\mathscr{d}'_i = |\mathscr{d}_i|$ for all r and i, where $|.|$ is the absolute value. Define $\bar{a}_{\mathbb{N}} = (\mathsf{tst}'_0, \mathsf{asgn}_0)\dots$ as follows: for all i, tst'_i contains $r \lhd s$ iff $\hat{\nu}'_i(r) \lhd \hat{\nu}'_i(s)$, where $\hat{\nu}'_i$ extends ν'_i with $\hat{\nu}'_i(\star) = \mathscr{d}'_i$ and $\hat{\nu}'_i(0) = 0$, and $r, s \in R \cup \{*, 0\}$ and $\lhd \in \{<, =\}$. Note that tst'_i is uniquely defined by $\nu'_i, \mathscr{d}'_i$; moreover, $(\nu'_i, \mathscr{d}'_i) \models \mathsf{tst}'_i$ and $(\mathsf{tst}_i, \mathsf{tst}'_i) \in \kappa$. Hence, $\bar{a}_{\mathbb{N}} \in K(\bar{a}_{\mathbb{Z}})$, and $\bar{\nu}_{\mathbb{N}}$ makes $\bar{a}_{\mathbb{N}}$ feasible in $(\mathbb{N}, \leq, 0)$.

Conversely, suppose there exists $\bar{a}_{\mathbb{N}} \in K(\bar{a}_{\mathbb{Z}})$ (which is in fact unique) feasible in $(\mathbb{N}, \leq, 0)$ by some $\bar{\nu}_{\mathbb{N}} = (\nu'_0, \mathscr{d}'_0)\dots$. Let $\bar{\nu}_{\mathbb{Z}} = (\nu_0, \mathscr{d}_0)\dots$ be a sequence of valuation-data pairs in $(\mathbb{Z}, \leq, 0)$ such that for all i: $\nu_i(r) = -\nu'_i(r)$ if $r < 0 \in \mathsf{tst}_i$, otherwise $\nu_i(r) = \nu'_i(r)$, for all $r \in R$; and $\mathscr{d}_i = -\mathscr{d}'_i$ if $\star < 0 \in \mathsf{tst}_i$, otherwise $\mathscr{d}_i = \mathscr{d}'_i$. Since $\bar{\nu}_{\mathbb{N}}$ makes $\bar{a}_{\mathbb{N}}$ feasible in $(\mathbb{N}, \leq, 0)$, and by definition of $\bar{\nu}_{\mathbb{Z}}$ and κ, $\bar{\nu}_{\mathbb{Z}}$ makes $\bar{a}_{\mathbb{Z}}$ feasible in $(\mathbb{Z}, \leq, 0)$. This concludes the proof that K satisfies property 1 of the definition of reducibility. ◁

Corollary 14. *Bounded synthesis is decidable for* $(\mathbb{Z}, \leq, 0)$.

Example: reducing $(\mathbb{N}^k, \leq^k, 0^k)$ to $(\mathbb{N}, \leq, 0)$

For $(n_1, ..., n_k), (m_1, ..., m_k) \in \mathbb{N}^k$, define $(n_1, ..., n_k) <^k (m_1, ..., m_k)$ iff for all $i \in \{1, ..., k\}$, $n_i < m_i$; similarly define $=^k$. The relation $<^k$ is a partial order. Recall that tests are defined to be satisfiable sets maximal wrt. inclusion, meaning that a test cannot have two registers unrelated, despite $<^k$ being a partial order. To model partial order in tests, define a predicate $?$: $a?b$ iff neither $a <^k b$ nor $a =^k b$ nor $b <^k a$, i.e., a and b are incomparable, for $a, b \in \mathbb{N}^k$. Let $(\mathbb{N}^k, \leq^k, 0^k)$ denote the data structure with elements $\mathbb{N}^k$, predicates $\{<^k, =^k, ?\}$, and constant 0^k.

Lemma 15. $(\mathbb{N}^k, \leq^k, 0^k)$ *reduces to* $(\mathbb{N}, \leq, 0)$.

▷**Proof.** Fix a set of registers R. Let $R_i = \{r^i \mid r \in R\}$ be an ith copy of R, for every $i \in \{1, ..., k\}$. We will construct a relation K between action words over R and action words over $R' = \bigcup_i R_i$ satisfying the definition of reducibility. To this end, define a relation κ between test-assignments over R and k-sequences of test-assignments over R': $\big((\mathsf{tst}, \mathsf{asgn}), (\mathsf{tst}^1, \mathsf{asgn}^1)\dots(\mathsf{tst}^k, \mathsf{asgn}^k)\big) \in \kappa$ iff

– $(r \lhd s) \in \mathsf{tst}$ implies $(r^i \lhd s^i) \in \mathsf{tst}^i$, for every $r^i, s^i \in R^i \cup \{\star, 0\}$, $\lhd \in \{<, =\}$, $i \in \{1, ..., k\}$;
– $(r\,?\,s) \in \mathsf{tst}$ implies there are i, j and different $\lhd, \bowtie \in \{<, =, >\}$ such that $r^i \lhd s^i \in \mathsf{tst}^i$ and $r^j \bowtie s^j \in \mathsf{tst}^j$; and
– $\mathsf{asgn}^i(r^i) = \mathsf{asgn}(r)^i$ for all $r \in R$, $i \in \{1, ..., k\}$.

Then, we define $(a_0 a_1 \dots, a^1_0 ... a^k_0 a^1_1 ... a^k_1 \dots) \in K$ iff $(a_j, a^1_j, ..., a^k_j) \in \kappa$ for all j, where every a_j is a test-assignment pair over R and every a^i_j is a test-assignment pair over R^i.

As K is a rational relation, it satisfies properties 2 and 3 of the definition of reducibility. We now show property 1. Fix a test-assignment word $\bar{a} = (\mathsf{tst}_0, \mathsf{asgn}_0)\dots$ over domain $(\mathbb{N}^k, \leq^k, 0^k)$ and registers R.

Suppose $\bar{a}$ is feasible in $(\mathbb{N}^k, \leq^k, 0^k)$ by some $\bar{\nu} = (\nu_0, \mathscr{d}_0)(\nu_1, \mathscr{d}_1)\dots$. We construct $\bar{a}' = (\mathsf{tst}^1_0, \mathsf{asgn}^1_0)...(\mathsf{tst}^k_0, \mathsf{asgn}^k_0)\,(\mathsf{tst}^1_1, \mathsf{asgn}^1_1)\dots$ over R' that is feasible in $(\mathbb{N}, \leq, 0)$ by some

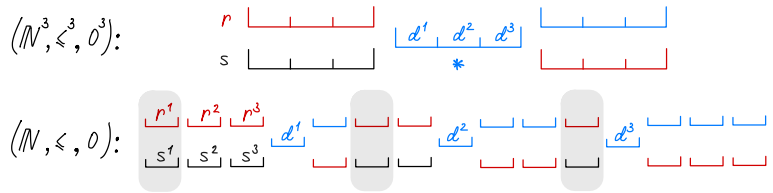

Figure 5 Let $R = \{r, s\}$ and consider $(\mathbb{N}^3, \leq^3, 0^3)$. Top: the registers r and s get updated in $(\mathbb{N}^3, \leq^3, 0^3)$: r takes the value of the input data $d \in \mathbb{N}^3$ while s copies the value of r. Bottom: in $(\mathbb{N}, \leq, 0)$, we have $R' = \{r^1, s^1, r^2, s^2, r^3, s^3\}$; the corresponding sequence of updates happens in three steps: $\{r^1, s^1\}$ are updated, then $\{r^2, s^2\}$, then $\{r^3, s^3\}$.

$\bar{\nu}' = (\nu_0^1, d_0^1)...(\nu_0^k, d_0^k)\,(\nu_1^1, d_1^1)\ldots$ and such that $\bar{a}' \in K(\bar{a})$. Define $\nu_j^i = \nu_j[i]$ and $d_j^i = d_j[i]$. Define tst_j^i to be a unique test s.t. $(\nu_j^i, d_j^i) \models \mathsf{tst}_j^i$ while asgn_j^i is as in the definition of κ. By construction (illustrated on Figure 5), $\bar{a}'$ is feasible by $\bar{\nu}'$ and $\bar{a}' \in K(\bar{a})$.

Suppose $\bar{a}' = a_0^1...a_0^k a_1^1...a_1^k \ldots \in K(\bar{a})$ is feasible by $\bar{\nu}' = (\nu_0^1, d_0^1)...(\nu_0^k, d_0^k)\,(\nu_1^1, d_1^1)\ldots$. Define $\bar{\nu} = (\nu_0, d_0)(\nu_1, d_1)\ldots$ where $\nu_j = \nu_j^1$ and $d_j = (d_j^1, ..., d_j^k)$, for all j. Again, (ν_j, d_j) satisfies every atom $r \lhd s$ of every tst_j by definition of κ, ν_j, d_j. Moreover, $\nu_{j+1} = \mathit{update}(\nu_j, d_j, \mathsf{asgn}_j)$. Therefore, $\bar{a}$ is feasible by $\bar{\nu}$.

$\lhd$

Corollary 16. *Bounded synthesis is decidable for* $(\mathbb{N}^k, \leq^k, 0^k)$.

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.