A Generic Solution to Register-bounded Synthesis with an Application to Discrete Orders (full version)

Léo Exibard
Reykjavik University, Iceland

Emmanuel Filiot
Université libre de Bruxelles, Belgium

Ayrat Khalimov
Université libre de Bruxelles, Belgium

Abstract

We study synthesis of reactive systems interacting with environments using an infinite data domain. A popular formalism for specifying and modelling such systems is register automata and transducers. They extend finite-state automata by adding registers to store data values and to compare the incoming data values against stored ones. Synthesis from nondeterministic or universal register automata is undecidable in general. However, its register-bounded variant, where additionally a bound on the number of registers in a sought transducer is given, is known to be decidable for universal register automata which can compare data for equality, i.e., for data domain $(\mathbb{N}, =)$. This paper extends the decidability border to the domain $(\mathbb{N}, <)$ of natural numbers with linear order. Our solution is generic: we define a sufficient condition on data domains (regular approximability) for decidability of register-bounded synthesis. The condition is satisfied by natural data domains like $(\mathbb{N}, <)$. It allows one to use simple language-theoretic arguments and avoid technical game-theoretic reasoning. Further, by defining a generic notion of reducibility between data domains, we show the decidability of synthesis in the domain $(\mathbb{N}^d, <^d)$ of tuples of numbers equipped with the component-wise partial order and in the domain $(\Sigma^*, \prec)$ of finite strings with the prefix relation.

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 Domains

Funding This work was supported by the Fonds de la Recherche Scientifique - F.R.S.-FNRS under the MIS project F451019F. Emmanuel Filiot is a senior research associate at F.R.S.-FNRS. Léo Exibard was supported by the project 'Mode(l)s of Verification and Monitorability (MoVeMnt)' (no. 217987-051) of the Icelandic Research Fund.

1 Introduction

Synthesis. Reactive synthesis aims at the automatic construction of an interactive 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 modelled as a language of desirable input-output sequences. The synthesis problem then asks to automatically construct a transducer whose input-output sequences belong to a given specification. Traditionally [30, 4], the inputs and outputs have been modelled as letters from a finite alphabet. This, however, limits the application of synthesis. Recently researchers have started investigating synthesis of systems working on data domains [12, 24, 15, 25, 2, 14].

Automata as specifications. In the finite-alphabet setting, specifications are usually given as logical formulas and a synthesiser performs a series of translations: first, from the formula to an automaton, then from the automaton to a game, and finally it searches for a winning strategy in the game. It is the second step, from automata to games, that captures the game-theoretic essence of synthesis, whereas the first step is an orthogonal problem of finding a convenient logical formalism. In the context of synthesis over data domains, this first step is problematic as there is no decidable, and expressive enough, logic having a corresponding automaton model. For that reason, in this paper we focus on the second step and use automata for specifications.

Register automata. A well-studied automata formalism for specifying and modelling data systems are register automata and transducers [22, 28, 23, 33]. Register automata extend classical finite-state automata to infinite alphabets $\mathbb{D}$ by introducing a finite number of *registers*. In each step, the automaton reads a data value $d \in \mathbb{D}$, compares it with the values held in its registers, then depending on this comparison it decides to store d 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 $\mathbb{D}^\omega$: it is accepting 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.

Universal register automata. Unlike classical finite-state automata, the expressive power of register automata depends on whether they are deterministic, nondeterministic, or universal (a.k.a. co-nondeterministic). Among these, universal register automata suit synthesis best. First, they can specify request-grant properties: every requested data shall be eventually outputted. This is a 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. Hence they allow for succinct conjunction of properties, which is desirable in synthesis as specifications usually consist of many independent properties. Finally, in the register-free setting universal automata are often used to obtain synthesis methods feasible in practice [26, 31, 17, 4].

Data domains with order. Another factor affecting expressivity of register automata is the data-comparison operators. Originally, register automata compared data for equality only, i.e., operated in data domain $(\mathbb{D}, =)$ [22]. This limits synthesis applications as we cannot specify priority arbiters [8] that should give a resource to a requesting process with the lowest ID. Such properties require data domains with linear order $<$ (in addition to $=$). Further, there are data domains with dense order, like $(\mathbb{Q}, <)$, and those with discrete order, like $(\mathbb{N}, <)$. The domain $(\mathbb{Q}, <)$ is well-suited for abstracting physical phenomena like changing temperature in a room. However, for abstracting hardware, the domain $(\mathbb{N}, <)$ suits better as it excludes Zeno-like behaviours (when a process ID gets infinitely closer to another ID but never reaches it). The domain $(\mathbb{N}, <)$ is also interesting from the theoretical point of view as it demands new proof techniques.

Known synthesis results for register automata. Already for $(\mathbb{D}, =)$, the synthesis problem of register transducers from universal register automata is undecidable [12, 15]. Decidability is recovered in the deterministic case [15, 14], but, as argued above, universal automata are more desirable in synthesis. To circumvent undecidability, the works [24, 15, 25] studied *register-bounded* synthesis: given a universal register automaton *and a bound k* on the number of transducer registers, return a k-register transducer realising the automaton or 'No' if no such transducer exists. They showed the decidability of register-bounded for $(\mathbb{D}, =)$, and it is not hard to adapt their techniques to $(\mathbb{Q}, <)$ and other oligomorphic domains [6], however the

	$(\mathbb{D},=)$	$(\mathbb{Q},<)$	$(\mathbb{N},<)$
DRA	✓[14]	✓[14]	✓[14]
NRA	✗[15]	✗	✗
URA	✗[15]	✗	✗

Table 1 Decidability of register-unconstrainted synthesis.

	$(\mathbb{D},=)$	$(\mathbb{Q},<)$	$(\mathbb{N},<)$
DRA	✓[24]	✓[13]	✓[THIS PAPER]
NRA	✗[15]	✗	✗
URA	✓[24]	✓[13]	✓[THIS PAPER]

Table 2 Decidability of register-bounded synthesis.

domain $(\mathbb{N},<)$ remained elusive. Tables 1 and 2 summarise known and new results, where DRA/NRA/URA stand for deterministic/nondeterministic/universal register automata.

Contributions. We prove that register-bounded synthesis is decidable for $(\mathbb{N},<)$ in time doubly exponential in the number of registers of the specification automaton and of the sought transducer. Our procedure is effective: it constructs a transducer if one exists. When the total number of registers is fixed, it is ExpTime-c, matching the complexity of classical (register-free) synthesis. This result generalises the works of [15, 25, 24] on $(\mathbb{D},=)$. We then extend the decidability boundary farther to include the domain $(\mathbb{N}^d,<^d)$ of tuples of naturals with the component-wise partial order, and the domain $(\Sigma^*,\prec)$ of strings with the prefix relation.

Technical contributions. Our proof technique is generic and greatly simplifies the task of proving new synthesis decidability results by removing the need to reason about synthesis alltogether. We now describe the technique in detail.

The key idea of existing approaches [24, 15, 25] is to reduce the register-bounded synthesis problem in a data domain to a two-player Church game with a finite alphabet and an ω-regular winning condition. In such a game, two players alternately play for an infinite number of rounds. Adam, modelling the environment, picks a test over the k registers describing how its input data compares with the current content of the registers of a sought transducer. Eve, modelling the system, picks a subset of the k registers, meant to store the data, and a register whose value is meant for output. No data are manipulated in the game. Infinite plays in the game induce infinite sequences of tests, assignments, and outputs over the k registers, called *action words*; they are over a *finite* alphabet. Action words are meant to abstract data words; an action word is *feasible* if there is at least one data word that satisfies all its tests and assignments. The reduction ensures that any strategy of Eve winning in the game can be converted into a k-register transducer realising the specification, and vice versa. To this end, the game winning condition declares a play to be won by Eve if all data words satisfying the action word induced by the play are accepted by the specification automaton. In particular, a play whose action word is *un*feasible is won by Eve as it does not correspond to any environment-system interaction in the data domain. In the case of $(\mathbb{D},=)$, such winning conditions are known to be ω-regular [24, 15, 25]. However, in $(\mathbb{N},<)$ the set of feasible action words is not ω-regular [14], and neither is the winning condition. Such winning conditions could be expressed by nondeterministic ωS automata [5], but games with such objectives are not known to be decidable, to the best of our knowledge.

To overcome the latter obstacle, we introduce the notion of ω-regularly approximable (*regapprox*) data domains. A regapprox data domain has an ω-regular over-approximation of the set of feasible action words that is exact on the lasso-shaped action words (of the form uv^ω). Thus, in regapprox domains the set of feasible lasso-shaped action words is ω-regular. This allows us to avoid dealing with non-ω-regularity and reduce synthesis to solving classic

ω-regular games. Our first technical contribution is the generic decidability result:

For regapprox domains, register-bounded synthesis from URA is decidable.

The procedure is constructive: for realisable specifications it outputs a transducer. Note that all oligomorphic domains [6], e.g. $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$, are regapprox, because their sets of feasible action words are ω-regular, so our result subsumes works [15, 25, 24]. For $(\mathbb{N}, <)$, we construct its over-approximation relying on the result [14], and then instantiate the theorem.

There are many domains with discrete order resembling $(\mathbb{N}, <)$: the domain $(\mathbb{Z}, <)$ of integers, the domain $(\mathbb{N}^d, <^d)$ of tuples of naturals with the component-wise partial order, and even the domain $(\Sigma^*, \prec)$ of strings with the prefix relation. To further simplify decidability proofs on these domains, we define a natural and generic notion of reducibility between data domains. Intuitively, a data domain $\mathbb{D}$ *reduces* to $\mathbb{D}'$ if there is a rational transduction that relates action words in $\mathbb{D}$ and $\mathbb{D}'$ while preserving feasibility. Our second technical contribution is the reduction result:

If $\mathbb{D}$ reduces to $\mathbb{D}'$, and $\mathbb{D}'$ is regapprox, then $\mathbb{D}$ is regapprox.

This implies that a synthesis procedure for $\mathbb{D}'$ can be used to solve synthesis in $\mathbb{D}$. We illustrate the technique by reducing to $(\mathbb{N}, <)$ the domains $(\mathbb{N}^d, <^d)$ and $(\Sigma^*, \prec)$. The reduction for $(\Sigma^*, \prec)$ relies on the work [10]. These reductions entail the decidability of register-bounded synthesis on these domains.

Related works. We already mentioned the works [24, 15, 25, 13] on synthesis of register transducers in domains $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$, and that our result generalises them for the case of URAs. The paper [14] studies *Church's synthesis* for DRA specifications, where a *data* strategy not necessarily with finitely-many states is sought. However, they show that considering register transducers is sufficient, with with the number of registers equal that of the specification automaton. Hence our register-bounded synthesis procedure for URAs can also be used to solve the Church's synthesis problem.

Another formalism for specifications of data systems is that of *variable automata* [20]. The paper [16] studies synthesis of symbolic transducers from specifications given in a fragment of nondeterministic variable automata. They solve synthesis for data domain $(\mathbb{Q}, <)$ and leave the domain $(\mathbb{N}, <)$ for future work. Variable automata are incomparable with register automata, and their particular fragment cannot express request-grant properties of arbiters that we believe is desirable in synthesis.

Our proof techniques resemble those from some works on satisfiability of data logics. Constraint LTL [11] extends Linear Temporal Logic (LTL) by atoms allowing one to compare data values within the horizon or pre-defined length. The satisfiability of this logic is decidable for data domains $(\mathbb{D}, =)$, $(\mathbb{Q}, <)$, $(\mathbb{N}, <)$ [11], and $(\Sigma^*, \prec)$ [10]. Their proof technique relies on the abstraction of data values at different moments by relations between each other. For the data domain $(\mathbb{N}, <)$, they additionally prove that considering lasso-shaped witnesses of satisfiability is sufficient. Our generic synthesis result uses a similar idea by defining regapprox domains. We note that formulas in Constraint LTL can always be translated into universal register automata (which are more expressive) [32]. Hence our approach can be used to solve register-bounded synthesis from Constraint LTL.

The papers [19, 27] suggest a sound/incomplete procedure to synthesis from Temporal Stream Logic. This logic extends LTL by adding the atoms that are either first-order predicate terms or are assignments of variables to a first-order function term. Similarly, transducers can test data using the predicate terms and update its values by the function terms. A transducer satisfies a specification if it does so under *every* interpretation of predicates and functions. It is possible to model domains like $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$ in their formalism, by encoding the axioms for $>$ and $=$ into specification. This would give a sound/*in*complete synthesis approach. Our approach is less general but retains the completeness.

More generally, our notion of regular approximation echoes a general idea common to verification techniques, for example of programs manipulating data variables (see, e.g., [21]), to abstract concrete behaviours by regular ones. When an over-approximation is used, it is guaranteed that if the abstract program satisfies some safety properties, so does the concrete program. This yields sound algorithm which are not necessarily complete. Here in the context of register automata, instead, we require that the over-approximation is exact on lasso-like executions, and show that this implies completeness (for the synthesis problem).

2 Synthesis Problem

Let $\mathbb{N} = \{0, 1, \dots\}$ denote the set of natural numbers including 0.

Data domain and data words. A *data domain* is a tuple $\mathcal{D} = (\mathbb{D}, P, C, c_0)$ consisting of an infinite countable set $\mathbb{D}$ of *data values*, a finite set P of interpreted *predicates* (predicate names with arities and their interpretations) which must contain the equality predicate $=$, a finite set $C \subset \mathbb{D}$ of *constants*, and a distinguished *initialiser* constant $c_0 \in C$. For example, $(\mathbb{N}, \{<, =\}, \{0\}, 0)$ is the data domain of natural numbers with the usual interpretation of $<, =$, and 0. In the tuple notation, we often omit the brackets, as well as the mention of $=$ and of c_0 when the initialiser constant is clear from the context. E.g., we write $(\mathbb{N}, <, 0)$ for $(\mathbb{N}, \{<, =\}, \{0\}, 0)$. Another familiar example is $(\mathbb{Z}, <, 0)$, which is the data domain of integers with the usual $<, =$, and 0. Throughout the paper we assume that the satisfiability problem of quantifier-free formulas built on the signature (P, C) is decidable in $\mathcal{D}$, and whenever we state complexity results, the satisfiability problem is additionally assumed to be decidable in PSpace. This is the case for all data domains considered in this paper. Finally, *data words* are infinite sequences $d_0 d_1 \dots \in \mathbb{D}^\omega$, and for two sets I and O and a language $L \subseteq (I \cdot O)^\omega$, we call I and O its *input* and *output* alphabets respectively.

Action words. Fix a data domain $\mathcal{D} = (\mathbb{D}, P, C, c_0)$ and a finite set R of elements called *registers*. A register *valuation* (over $\mathcal{D}$) is a mapping $\nu : R \to \mathbb{D}$. Given a valuation ν, a variable x (not necessarily in R), and a data value $d \in \mathbb{D}$, define $\nu[x \leftarrow d]$ to be the valuation $R \cup \{x\} \to \mathbb{D}$ that maps x to d and every $r \in R \setminus \{x\}$ to $\nu(r)$. We extend this notation to any finite set $A = \{a_1, \dots, a_n\}$ by letting $\nu[A \leftarrow d] = \nu[a_1 \leftarrow d] \dots [a_n \leftarrow d]$.

A *test* (over $\mathcal{D}$) is a conjunction ($\wedge$) of distinct literals over predicates P and constants C, encoded as a set of literals $p(x_1, \dots, x_a)$ and $\neg p(x_1, \dots, x_a)$, where $p \in P$, a is the arity of p and $x_1, \dots, x_a \in R \cup C \cup \{\star\}$. The symbol $\star$ is a fresh symbol used as a placeholder for incoming data values. By convention, $\wedge\varnothing = \top$, and the empty set encodes the test that is always true. Depending on the context, we use the formula or set notation. A register valuation $\nu : R \to \mathbb{D}$ and data value $d \in \mathbb{D}$ *satisfy* a test φ, written $\nu, d \models \varphi$, if $\nu[\star \leftarrow d]$ satisfies φ, where predicates and constants are interpreted in the data domain $\mathcal{D}$. A test φ is *maximal* if it specifies the relation between all variables and constants wrt. the predicates, i.e. it is a maximally consistent conjunction of literals: $\varphi = \bigwedge_{\substack{p \in P \\ p \text{ of arity } r}} \bigwedge_{\substack{x_1, \dots, x_r \\ \in R \cup C \cup \{\star\}}} l_{p, x_1, \dots, x_r}$, where $l_{p, x_1, \dots, x_r} \in \{p(x_1, \dots, x_r), \neg p(x_1, \dots, x_r)\}$. Maximal tests are mutually exclusive: a given valuation cannot satisfy simultaneously two of them. Observe that a test is equivalent to a (possibly exponential) disjunction of maximal ones. Let $\mathsf{Tst}_R^{\mathcal{D}}$ denote the set of all possible tests over registers R in domain $\mathcal{D}$, and $\mathsf{MTst}_R^{\mathcal{D}} \subset \mathsf{Tst}_R^{\mathcal{D}}$ the subset of maximal ones.

▶ Example. Consider domain $(\mathbb{N}, <, 0)$ and $R = \{r\}$. Atomic formulas are $r < \star$, $\star = r$, $r < 0$, $\star = 0$, etc. The test $0 < r \wedge r < \star$ specifies that the content of register r is strictly positive and that the incoming data is greater than it. It is not maximal, since it does not contain the atoms $0 < \star$, $\neg(\star = r)$, $\neg(\star = 0)$, $\neg(r = 0)$. For readability, we write $0 < r < \star$.

An *assignment* is a set $\mathsf{asgn} \subseteq R$ of registers meant to store the current input data value. Let $\mathsf{Asgn}_R = 2^R$ denote the set of all possible assignments. An *action* is a pair $(\mathsf{tst}, \mathsf{asgn}) \in \mathsf{Tst}_R \times \mathsf{Asgn}_R$. We now describe how valuations are updated: given a valuation ν, a data value d, a test tst and an assignment asgn, we say that the valuation ν' is the *successor* of ν following action $(\mathsf{tst}, \mathsf{asgn})$ on reading d, written $\nu \xrightarrow{d,\mathsf{tst},\mathsf{asgn}} \nu'$, if the data value satisfies the test, i.e. $\nu, d \models \mathsf{tst}$, and $\nu' = \nu[\mathsf{asgn} \leftarrow d]$.

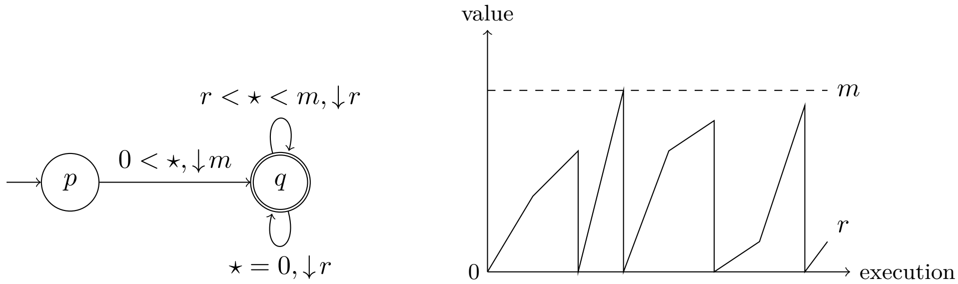
An *automaton action word*, or simply *action word*, is an infinite sequence of actions $\bar{a} = (\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1) \ldots \in (\mathsf{Tst}_R \times \mathsf{Asgn}_R)^\omega$. It is *feasible* by a sequence of valuation-data pairs $(\nu_0, d_0)(\nu_1, d_1) \ldots$ if $\nu_0 : r \in R \mapsto c_0$, i.e. ν_0 maps every $r \in R$ to c_0, and for all i: $\nu_i \xrightarrow{d_i,\mathsf{tst}_i,\mathsf{asgn}_i} \nu_{i+1}$. We then say that the data word $d_0 d_1 \ldots$ is *compatible* with $\bar{a}$. Let $\mathsf{AW}_R^{\mathcal{D}}$ denote the set of action words over R in $\mathcal{D}$, and $\mathsf{FEAS}_R^{\mathcal{D}}$ the subset of feasible ones. We may write either AW_R, or $\mathsf{AW}^{\mathcal{D}}$ or just AW when $\mathcal{D}$, R or both are clear from the context, similarly for FEAS.

▶ Example. Consider domain $(\mathbb{N}, <, 0)$ and $R = \{r\}$. For $r \in R$, the assignment $\{r\}$ is denoted $\downarrow r$. The action word $(0 < \star, \downarrow r)(\star < r, \downarrow r)^\omega$ is unfeasible in $(\mathbb{N}, <, 0)$, because it requires having an infinite chain of strictly decreasing values, which is not possible since $\mathbb{N}$ is well-founded. The same action word can be interpreted in $(\mathbb{Z}, <, 0)$ and in $(\mathbb{Q}, <, 0)$ and there it is feasible, as well as in $(\mathbb{Q}_+, <, 0)$ since $\mathbb{Q}_+$ is dense.

Register automata. A register automaton over data domain $\mathcal{D}$ 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*, $\delta \subseteq Q \times \mathsf{Tst}_R \times \mathsf{Asgn}_R \times Q$ 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 $(p, \nu) \in Q \times \mathbb{D}^R$; it is *initial* if $p = q_0$ and $\nu : r \in R \mapsto c_0$. The configuration (q, ν') is a *successor* of (p, ν) on reading data value $d \in \mathbb{D}$ and taking transition $p' \xrightarrow{\mathsf{tst},\mathsf{asgn}} q' \in \delta$, written $(p, \nu) \xrightarrow[S]{d,\mathsf{tst},\mathsf{asgn}} (q, \nu')$ or simply $(p, \nu) \xrightarrow[S]{d} (q, \nu')$, if $p = p'$, $q = q'$ and $\nu \xrightarrow{d,\mathsf{tst},\mathsf{asgn}} \nu'$, i.e. $\nu, d \models \mathsf{tst}$ and $\nu' = \nu[\mathsf{asgn} \leftarrow d]$.

A *run* of S over a data word $d_0 d_1 \ldots$ is a sequence of configurations $\rho = (q_0, \nu_0)(q_1, \nu_1) \ldots$ such that (q_0, ν_0) is initial and for every i, (q_{i+1}, ν_{i+1}) is a successor of (q_i, ν_i) on reading d_i, on taking some transition $q_i \xrightarrow{\mathsf{tst}_i,\mathsf{asgn}_i} q_{i+1} \in \delta$. We then say that the automaton action word $(\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1) \ldots$ *labels* ρ; note that it is feasible by $\nu_0 d_0 \nu_1 d_1 \ldots$. The run ρ is *accepting* if the maximal priority appearing infinitely often in $\alpha(q_0)\alpha(q_1) \ldots$ is even, otherwise it is *rejecting*. A data word may have several runs of S. For *universal* register automata, abbreviated URA, a word is *accepted* if all its runs are accepting; for *nondeterministic* automata, there should be at least one accepting run. The set of all data words over $\mathcal{D}$ accepted by S is called the *language* of S and denoted $L(S)$. We may write $L_{\mathcal{D}}(S)$ to emphasise that $L(S)$ is defined over $\mathcal{D}$.

A *finite (parity) 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. Such automata operate on words from Σ^ω.



(a) A register automaton over $(\mathbb{N}, <, 0)$ (state q is accepting).

(b) An example of the sequence of values taken by registers r and m along the run.

Figure 1 A register automaton whose action words do not form an ω-regular language.

Syntactical language of a register automaton. A register automaton $S = (Q, q_0, R, \delta, \alpha)$ can be treated syntactically, it then induces a universal finite automaton $S_{synt} = (\Sigma, Q, q_0, \delta, \alpha)$ with $\Sigma = \mathsf{Tst}_R \times \mathsf{Asgn}_R$. Note that since S_{synt} is universal, words that have no run are accepted. Notice that the language of S_{synt} may contain action words which are not feasible.

▶ Example. Consider the automaton of Figure 1a. Its syntactical language is

$$(0 < \star, \downarrow m)\big((r < \star < m, \downarrow r) \mid (0 = \star, \downarrow r)\big)^{\omega}$$

which includes not only feasible but also unfeasible action words, e.g. $(0 < \star, \downarrow m)(r < \star < m, \downarrow r)^{\omega}$. The feasible accepted action words have the form

$$(0 < \star, \downarrow m) \prod_{i=1}^{\infty} \big((r < \star < m, \downarrow r)^{n_i} (0 = \star, \downarrow r)\big)$$

such that the numbers $(n_i)_i$ are uniformly bounded by some value; the bound corresponds to the first read data value. This language is not ω-regular but an ωB-language [5].

Register transducers. A *k-register transducer* is a tuple $T = (Q, q_0, R, \delta)$, where Q, q, R ($|R| = k$) are as in automata but $\delta : Q \times \mathsf{MTst} \to \mathsf{Asgn} \times R \times Q$. Note that δ is a total function; moreover, since we restrict to maximal tests, exactly one test holds per incoming data value, so the transducers are deterministic and complete. A configuration is a pair $(p, \nu) \in Q \times \mathbb{D}^R$. From configuration (p, ν), on reading $\mathscr{d} \in \mathbb{D}$, the transducer takes the unique transition $p \xrightarrow{\mathsf{tst}, \mathsf{asgn} | r} q$ such that $\nu, \mathscr{d} \models \mathsf{tst}$, updates its configuration to (q, ν') where $\nu \xrightarrow{\mathscr{d}, \mathsf{tst}, \mathsf{asgn}} \nu'$, and outputs the value $\nu'(r)$. Note that the output is produced *after* assignment. We then write $(p, \nu) \xrightarrow[T]{\mathscr{d}, \mathsf{tst}, \mathsf{asgn} | r, \nu'(r)} (q, \nu')$, or simply $(p, \nu) \xrightarrow[T]{\mathscr{d} | \nu'(r)} (q, \nu')$. A *run* of T on an input data word $\mathscr{d}^{\mathsf{i}}_0 \mathscr{d}^{\mathsf{i}}_1 \ldots$ is a sequence $(q_0, \nu_0)(q_1, \nu_1)\ldots$ such that (q_0, ν_0) is initial and for all $i \geq 0$, $(q_i, \nu_i) \xrightarrow{\mathscr{d}^{\mathsf{i}}_i, \mathsf{tst}_i, \mathsf{asgn}_i | r_i, \mathscr{d}^{\mathsf{o}}_i} (q_{i+1}, \nu_{i+1})$ for some unique $\mathscr{d}^{\mathsf{o}}_i \in \mathbb{D}$. The sequence $\mathscr{d}^{\mathsf{o}}_0 \mathscr{d}^{\mathsf{o}}_1 \ldots$ is the *output word* of T on reading $\mathscr{d}^{\mathsf{i}}_0 \mathscr{d}^{\mathsf{i}}_1 \ldots$; since the transducers are deterministic and have a run on every input word, the output word is uniquely defined. The sequence $\mathscr{d}^{\mathsf{i}}_0 \mathscr{d}^{\mathsf{o}}_0 \mathscr{d}^{\mathsf{i}}_1 \mathscr{d}^{\mathsf{o}}_1 \ldots$ is called the *input-output word.* We then say that the *transducer action word* $\mathsf{tst}_0(\mathsf{asgn}_0, r_0)\mathsf{tst}_1(\mathsf{asgn}_1, r_1) \ldots \in \big(\mathsf{MTst} \cdot (\mathsf{Asgn} \times R)\big)^{\omega}$ is *feasible* by $(\nu_0, \mathscr{d}^{\mathsf{i}}_0, \mathscr{d}^{\mathsf{o}}_0)(\nu_1, \mathscr{d}^{\mathsf{i}}_1, \mathscr{d}^{\mathsf{o}}_1)\ldots$. It is naturally associated with the automaton action word $(\mathsf{tst}_0, \mathsf{asgn}_0)(\star = r_0, \varnothing)(\mathsf{tst}_1, \mathsf{asgn}_1)(\star = r_1, \varnothing)\ldots$, which is then feasible by $\nu_0 \mathscr{d}^{\mathsf{i}}_0 \nu_1 \mathscr{d}^{\mathsf{o}}_0 \nu_1 \mathscr{d}^{\mathsf{i}}_1 \nu_2 \mathscr{d}^{\mathsf{o}}_1 \ldots$. The set of all transducer action words over R in data domain $\mathcal{D}$ is denoted by $\mathsf{TW}^{\mathcal{D}}_R$. The *language* $L(T)$ consists of all input-output words of T.

A *finite transducer* is a standard Mealy machine: it is a tuple $(\Sigma, \Gamma, Q, q_0, \delta)$, where Σ and Γ are finite input and output alphabets, $\delta : Q \times \Sigma \to \Gamma \times Q$, and the definition of language is standard. Treating a register transducer T syntactically gives a finite transducer denoted T_{synt} of the same structure as T with $\Sigma = \mathsf{MTst}_R$ and $\Gamma = \mathsf{Asgn}_R \times R$.

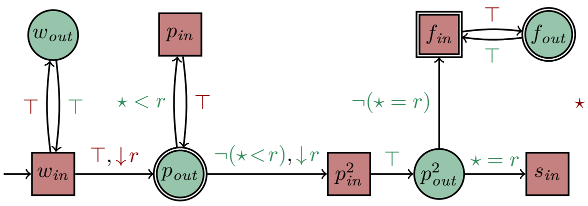
Synthesis problem. Fix a data domain $(\mathbb{D}, P, C, c_0)$. A register transducer T *realises* a register automaton S if $L(T) \subseteq L(S)$. The *register-bounded synthesis problem* is:

- input: $k \in \mathbb{N}$ and a URA S;
- output: yes iff there exists a k-register transducer which realises S.

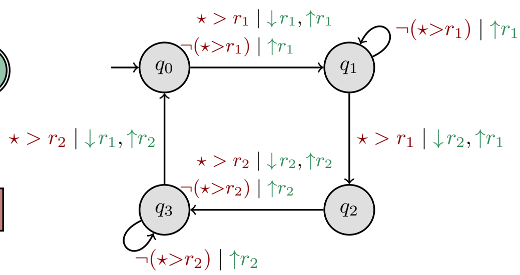
In this paper, when the synthesis problem is decidable, we are able to synthesise, i.e., effectively construct, a transducer realising the specification. We now make two remarks. First, notice that the number of transducer states is finite but unconstrained. Thus, register-bounded synthesis generalises classical register-free synthesis from (data-free) ω-regular specifications. Second, observe that transducers are complete, and therefore produce an ouptput word on every input word. Thus, a specification for which some input words do not have an associated output word is unrealisable. It is known that in the finite-alphabet case, the refined synthesis problem of good-enough synthesis [1], which requires a transducer to react only to inputs that belong to the domain of the specification, is still decidable. However, the good-enough register-bounded synthesis is undecidable [13, Chapter 8].

▶ Example. We illustrate the synthesis problem by describing a specification, its URA, and a register transducer realising it.

Let us start with the specification of priority arbiters. Such an arbiter reads an ID of a process requesting the resource, and outputs an ID of a process to whom the resource is granted. The specification requires that every requesting process is either acknowledged consecutively twice on the output, or this is done for a process of higher ID. We model the specification using the URA over $(\mathbb{N}, <, 0)$ with a single register from Figure 2a.



(a) URA for the priority-arbiter specification.



(b) Register transducer realising the specification.

Figure 2 A URA specification and a transducer implementing it.

The automaton reads words interleaving between arbiter data input and output, so its states are partitioned into box states (for reading input) and circle states (for reading output). The double-circle states are rejecting and can be visited only finitely often. Thus, a run looping in wait states w_{in} and w_{out} is accepting. Branching is universal, hence some run always loops around w_{in} and w_{out}. On reading an ID of a requesting process, a copy of the automaton moves from w_{in} to a pending state p_{out} while storing the ID into register r. It stays in states p_{out} and p_{in} as long as the request is not acknowledged, and such an infinite run is rejecting. If the request is eventually acknowledged (transitions from p_{out} to a sink state s_{in}), the run dies, so it is accepting. If a run reaches the failure state f_{in}, it is rejecting.

Figure 2b depicts a transducer with two registers r_1 and r_2 realising the above specification. On the left of the vertical bar are the tests over the inputs received by the transducer (in red), and on the right is the output action performed by the transducer (in green). For example, from state q_0 to q_1, if the input data $\mathscr{d}$ is larger than the data stored in register r_1, the transducer stores it into r_1, and outputs the content of r_1. The transducer uses one register to store the maximal value seen so far, while outputting the content of the other register, and the roles of these registers interchange as the transducer transits along the states. Thus, the instance of register-bounded synthesis with the described URA and $k = 2$ has a positive answer. However, when $k = 1$ the answer is negative.

3 Sufficient Condition for Decidable Synthesis for URA

In this section, we first show a reduction from register-bounded synthesis to (register-free) finite-alphabet synthesis. In the following, we fix a data domain $\mathcal{D}$. Given a specification S (as a URA over $\mathcal{D}$) and a bound k, we show how to construct a finite-alphabet specification $W^{\mathsf{F}}_{S,k}$ on action words over k registers, which is realisable by a finite-alphabet transducer iff S is realisable by a k-register transducer (Lemma 1). The main idea is to see the actions of the URA and of the sought k-register transducer as finite-alphabet letters. In particular, the specification $W^{\mathsf{F}}_{S,k}$ accepts a transducer action word $\bar{a}_k$ iff every action word $\bar{a}_S$ of the specification S, such that both $\bar{a}_k$ and $\bar{a}_S$ are feasible by the same data word, is accepted by S_{synt}. One can compose automata and transducer action words through a form of parallel product, which allows to talk about their joint feasibility. Then, in general, $W^{\mathsf{F}}_{S,k}$ is not necessarily ω-regular, and in a second step, we provide sufficient conditions on the data domain making synthesis wrt. $W^{\mathsf{F}}_{S,k}$ decidable, namely, that it can be under-approximated by an ω-regular language which coincides with $W^{f}_{S,k}$ over lasso words (Section 3.1). We obtain a general decidability result for data domains having this property (Theorem 4). We then instantiate this result for data domain $(\mathbb{N}, <, 0)$ (Section 3.2).

In the following, we fix a URA S with registers R_S and a disjoint set R_k consisting of k registers, and let $R = R_S \uplus R_k$. Given a transducer action word $\bar{a}_k = \mathsf{tst}^k_0\,(\mathsf{asgn}^k_0, r^k_0)\ldots \in \mathsf{TW}^{\mathcal{D}}_{R_k}$ and an automaton action word $\bar{a}_S = (\mathsf{tst}^{S\mathfrak{i}}_0, \mathsf{asgn}^{S\mathfrak{i}}_0)(\mathsf{tst}^{S\mathfrak{o}}_0, \mathsf{asgn}^{S\mathfrak{i}}_0)\ldots \in \mathsf{AW}^{\mathcal{D}}_{R_S}$, the product $\bar{a}_k \otimes \bar{a}_S$ of $\bar{a}_k$ and $\bar{a}_S$ is the automaton action word over registers R defined as $(\mathsf{tst}^k_0 \wedge \mathsf{tst}^{S\mathfrak{i}}_0, \mathsf{asgn}^k_0 \cup \mathsf{asgn}^{S\mathfrak{i}}_0)((\star = r^k_0) \wedge \mathsf{tst}^{S\mathfrak{o}}_0, \mathsf{asgn}^{S\mathfrak{i}}_0)\ldots$, which is essentially the parallel product of $\bar{a}_S$ and of the automaton word associated with $\bar{a}_k$.

We now show how to abstract a data specification given as URA S with registers R_S by a *finite-alphabet* specification over k-register transducer action words. Let $\mathsf{FEAS}^{\mathcal{D}}_R$ be the set of automata action words over R feasible in $\mathcal{D}$, then we define

$$W^{\mathsf{F}}_{S,k} = \left\{\bar{a}_k \in \mathsf{TW}_{R_k} \mid \forall \bar{a}_S \in \mathsf{AW}_{R_S} : \bar{a}_k \otimes \bar{a}_S \in \mathsf{FEAS}^{\mathcal{D}}_R \;\Rightarrow\; \bar{a}_S \in L(S_{synt})\right\}.$$

Thus, $W^{\mathsf{F}}_{S,k}$ *rejects* a feasible transducer action word $\bar{a}_k$ iff there is an automaton action word $\bar{a}_S$ feasible by the same data word as $\bar{a}_k$ and rejected by S.

▶ **Lemma 1.** *These two are equivalent:*

- *a URA S is realisable by a k-register transducer,*
- *$W^{\mathsf{F}}_{S,k}$ is realisable (by a finite-alphabet transducer).*

Proof. $\Rightarrow$: Assume that S is realisable by a register transducer T, i.e. $L_{\mathcal{D}}(T) \subseteq L_{\mathcal{D}}(S)$. Let $\bar{a}_k \in L(T_{synt})$, and let $\bar{a}_S \in \mathsf{AW}_{R_S}$ such that $\bar{a}_k \otimes \bar{a}_S \in \mathsf{FEAS}^{\mathcal{D}}_R$. Then, $\bar{a}_k \otimes \bar{a}_S$ is feasible by some input-output data word $w = \mathscr{d}^{\mathfrak{i}}_0 \mathscr{d}^{\mathfrak{o}}_0 \mathscr{d}^{\mathfrak{i}}_1 \mathscr{d}^{\mathfrak{o}}_1 \ldots$. By definition of the product, both $\bar{a}_k$ and $\bar{a}_S$ are feasible by w. Since $L_{\mathcal{D}}(T) \subseteq L_{\mathcal{D}}(S)$, if $\bar{a}_S$ labels a run of S on w, it means that it is accepting otherwise $w \notin L_{\mathcal{D}}(S)$ since S is a universal automaton. Thus, $\bar{a}_S \in L(S_{synt})$.

$\Leftarrow$: Conversely, assume that $W^{\mathsf{F}}_{S,k}$ is realisable by some finite transducer M, and let T be the associated register transducer, i.e. such that $T_{synt} = M$. Let $w \in L_{\mathcal{D}}(T)$ and let $\bar{a}_k$ be the action word labelling the run of T on w. Let $\bar{a}_S$ be an action word labelling a run of S on w if it exists (it might be that w is accepted by S by having no run on it). Then, $\bar{a}_k \otimes \bar{a}_S$ is feasible by w. By definition of $W^{\mathsf{F}}_{S,k}$, it means that $\bar{a}_S \in L(S_{synt})$, so $\bar{a}_S$ labels an accepting run of S on w. Overall, all runs of S on w are accepting, so $w \in L_{\mathcal{D}}(S)$. Thus, $L_{\mathcal{D}}(T) \subseteq L_{\mathcal{D}}(S)$, i.e. T realises S. ◀

3.1 General Decidability Result

In $(\mathbb{N}, <, 0)$, $W^{\mathsf{F}}_{S,k}$ is not ω-regular in general. To overcome this obstacle, we define the notion of ω-regularly approximable data domains. Such domains have an ω-regular equi-realisable subset of $W^{\mathsf{F}}_{S,k}$.

Let $lasso_R$ be the set of lasso-shaped[1] action words over a given set of registers R; we write *lasso* when R is clear. A data domain $\mathbb{D}$ is *ω-regularly approximable* (*regapprox*) if for every R there exists an ω-regular language $\mathsf{QFEAS}_R \subseteq (\mathsf{Tst}_R \times \mathsf{Asgn}_R)^\omega$ satisfying

$$\mathsf{QFEAS}_R \cap lasso_R \;\subseteq\; \mathsf{FEAS}_R \;\subseteq\; \mathsf{QFEAS}_R$$

and recognisable by a nondeterministic Büchi automaton that can be effectively constructed given R. The definition implies that FEAS_R and QFEAS_R coincide on lasso words. Such a set QFEAS_R is called *regular approximation* and written as QFEAS when R is clear.

▶ Example. The data domains $(\mathbb{D}, =)$ and $(\mathbb{Q}, <)$ are regapprox because their sets FEAS_R for every R are ω-regular, so there is no need to approximate them. On these domains, to check whether a given action word is feasible, one can track the relations between the registers and check if the read tests are consistent with these relations. For instance, if $r_1 < r_2$ but we read the test $* = r_1 = r_2$, then the action word is unfeasible.

The domain $(\mathbb{N}, <, 0)$ is also regapprox. Here, it is not sufficient to track the relations between the registers. We also need to ensure that between any two stored data values only a bounded number of different values is inserted along the action word. (Recall the example on page 7 with Figure 1a.) However, when an action word is lasso-shaped, it suffices to check the absence of an *infinite* number of such insertions. The latter can be checked by an ω-regular automaton, which allows for proving the regapproximability of $(\mathbb{N}, <, 0)$.

Finally, consider the data domain $(\mathbb{N}, \{S, =\}, \{0\}, 0)$, where S is the successor relation, i.e. $S(a, b)$ holds iff $a = b + 1$. This domain is not regapprox. Intuitively, this is because the domain allows for counting, which enables non ω-regular phenomena even in lasso words. We prove this by contradiction. Consider the following ω-regular language of action words over a single register r:

$$L \;=\; \left\{ \big(S(*, r), \downarrow r\big)^n \big(S(r, *), \downarrow r\big)^m \big(* = 0 = r, \varnothing\big)^\omega \mid n, m \in \mathbb{N} \right\},$$

i.e. the value in r is incremented n times, then decremented m times, then compared to zero and not updated. L contains feasible as well as unfeasible action words. Every feasible word of L has $n = m$, hence $\mathsf{FEAS} \cap L$ is not ω-regular. Moreover, every word of L is a lasso, thus $L \cap lasso = L$. Let us assume that the data domain is regapprox, witnessed by QFEAS for $R = \{r\}$. Since $\mathsf{QFEAS} \cap lasso = \mathsf{FEAS} \cap lasso$ by definition, we get

$$\mathsf{QFEAS} \cap L = \mathsf{QFEAS} \cap lasso \cap L = \mathsf{FEAS} \cap lasso \cap L = \mathsf{FEAS} \cap L.$$

[1] A word w is *lasso-shaped* (or regular, or ultimately periodic) if it is of the form $w = uv^\omega$ for some finite words u and v.

The language $\mathsf{QFEAS} \cap L$ is ω-regular, but $\mathsf{FEAS} \cap L$ is not. Contradiction. Therefore $(\mathbb{N}, \{S, =\}, \{0\}, 0)$ is not regapprox. ◀

Given a URA S with registers R_S and k, we define

$$W^{\mathsf{QF}}_{S,k} = \big\{\bar{a}_k \mid \forall \bar{a}_S \colon \bar{a}_k \otimes \bar{a}_S \in \mathsf{QFEAS}_R \;\Rightarrow\; \bar{a}_S \in L(S_{synt})\big\},$$

where $R = R_S \uplus R_k$. The definition of $W^{\mathsf{QF}}_{S,k}$ differs from $W^{\mathsf{F}}_{S,k}$ only in using QFEAS_R instead of FEAS_R. Since $\mathsf{FEAS}_R \subseteq \mathsf{QFEAS}_R$, we have $W^{\mathsf{QF}}_{S,k} \subseteq W^{\mathsf{F}}_{S,k}$.

We now show that $W^{\mathsf{QF}}_{S,k}$ is ω-regular (which essentially follows from ω-regularity of QFEAS and S_{synt}), and estimate the size of an automaton recognising $W^{\mathsf{QF}}_{S,k}$ and the time needed to construct it. For that 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 a constant $\kappa \in \mathbb{N}$. When $poly$, exp, and $2exp$ are used with several arguments, the maximal among them shall be taken for t. The construction and complexity analysis rely on standard automata techniques; we refer to Appendix A.1 for details.

▶ **Lemma 2.** *Let S be a URA and let $k \geq 1$. Then, $W^{\mathsf{QF}}_{S,k}$ is ω-regular. Moreover, $W^{\mathsf{QF}}_{S,k}$ is recognisable by a universal co-Büchi automaton with $O(2^k Nnc)$ many states that can be constructed in time $poly(N, n, exp(r, k))$, where n, r, and c are the number of states, registers, and priorities in S, and N is the number of states in a nondeterministic Büchi automaton recognising $\mathsf{QFEAS}_{R_S \uplus R_k}$.*

We now prove that $W^{\mathsf{F}}_{S,k}$ and $W^{\mathsf{QF}}_{S,k}$ are equi-realisable. For ω-regular specifications (like $W^{\mathsf{QF}}_{S,k}$) there is no distinction between realisability by finite- and infinite-state transducers [7]. This is not known for $W^{\mathsf{F}}_{S,k}$ specifications over domains such as $(\mathbb{N}, <, 0)$; we leave this question for future work, and in this paper focus on realisability by *finite-state* transducers.

▶ **Lemma 3.** *$W^{\mathsf{F}}_{S,k}$ is realisable by a finite-state transducer iff $W^{\mathsf{QF}}_{S,k}$ is realisable by a finite-state transducer.*

Proof. Direction $\Leftarrow$ follows from the inclusion $\mathsf{FEAS} \subseteq \mathsf{QFEAS}$, which implies $W^{\mathsf{QF}}_{S,k} \subseteq W^{\mathsf{F}}_{S,k}$. Consider direction $\Rightarrow$. Let T be a finite-state transducer that T does not realise $W^{\mathsf{QF}}_{S,k}$. We show that T does not realise $W^{\mathsf{F}}_{S,k}$ either. First, we have that $L(T) \not\subseteq W^{\mathsf{QF}}_{S,k}$, so the language $\{\bar{a}_k \otimes \bar{a}_S \in \mathsf{AW}^{\mathcal{D}}_R \mid \bar{a}_k \in L(T) \wedge \bar{a}_k \otimes \bar{a}_S \in \mathsf{QFEAS} \wedge \bar{a}_S \notin L(S_{synt})\}$ is nonempty. Since QFEAS and $L(S_{synt})$ are ω-regular, and since T is a finite-state transducer, this language is ω-regular. Thus, it contains a lasso-shaped word $\bar{a}_k \otimes \bar{a}_S$; by definition of the product, both $\bar{a}_k$ and $\bar{a}_S$ are then lasso-shaped. Since $\mathsf{QFEAS} \cap \mathit{lasso} \subseteq \mathsf{FEAS}$, we get that $\bar{a}_S$ is feasible, i.e. $\bar{a}_k \otimes \bar{a}_S \in \{\bar{a}_k \otimes \bar{a}_S \mid \bar{a}_k \in L(T) \wedge \bar{a}_k \otimes \bar{a}_S \in \mathsf{FEAS} \wedge \bar{a}_S \notin L(S_{synt})\}$, which implies that $L(T) \not\subseteq W^{\mathsf{F}}_{S,k}$: T does not realise $W^{\mathsf{F}}_{S,k}$. ◀

We are now able to prove the main result of this paper.

▶ **Theorem 4.** *Let $\mathcal{D}$ be a regapprox data domain such that for every set of registers R, one can construct a nondeterministic Büchi automaton with n_{QF} states recognising QFEAS_R in time $f(|R|)$ for some function f. Then:*

- *register-bounded synthesis for URAs over $\mathcal{D}$ is decidable in time $exp(exp(k, r), n_{\mathsf{QF}}, n, c) + f(k + r)$, where n is the number of states of the URA, c its number of priorities, r its number of registers, k is the number of transducer registers. It is* ExpTime*-c for fixed r and k.*
- *For every positive instance of the register-bounded synthesis problem, one can construct, within the same time complexities, a register transducer realising the specification.*

Proof. Lemmas 1,2,3 reduce register-bounded synthesis to (finite-alphabet) synthesis for the ω-regular specification $W^{\mathsf{QF}}_{S,k}$. Since synthesis wrt. to ω-regular specifications is decidable, we get the decidability part of the theorem. Let us now study the complexity. Let R_S be the set of r registers of the URA and R_k be a disjoint set of k registers. First, one needs to construct an automaton recognising $\mathsf{QFEAS}_{R_S \cup R_k}$. This is done by assumption in time $f(k+r)$. Then, one can apply Lemma 2 and get that $W^{\mathsf{QF}}_{S,k}$ can be recognised by universal co-Büchi automaton A with $O(2^k n_{qf} nc)$ states, which can be constructed in time $poly(n_{qf}, n, exp(r,k))$. A universal co-Büchi automaton with m states can be determinised into a parity automaton with $exp(m)$ states and $poly(m)$ priorities (see e.g. [29]). Recall that the alphabet of A is $\mathsf{Tst}_k \cup (\mathsf{Asgn}_k \times R_k)$. Hence by determinising A, and seeing it as a two-player game arena, we get a parity game with $exp(k)$ edges (corresponding to the actions of Adam and Eve), $exp(exp(k), n_{\mathsf{QF}}, n, c))$ states, and $poly(exp(k), n_{\mathsf{QF}}, n, c))$ 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. [9]), giving the overall time complexity $exp(exp(k), n_{\mathsf{QF}}, n, c))$ for solving the game. If we sum this to the complexity of constructing an automaton for $W^{\mathsf{QF}}_{S,k}$ plus the complexity for construction an automaton for QFEAS, we get $exp(exp(k), n_{\mathsf{QF}}, n, c)) + poly(n_{\mathsf{QF}}, n, exp(r,k)) + f(r+k)$, which is $exp(exp(k,r), n_{\mathsf{QF}}, n, c)) + f(r+k)$. If both r and k are fixed, then $exp(k,r)$ and $f(r+k)$ are constants, so the complexity is exponential only. It is folklore that the hardness holds in the register-free setting (for $r = k = 0$). See for example [18, Proposition 6] for a proof in the finite word setting over a finite alphabet (which straightforwardly generalises to infinite words). There, the proof is done for nondeterministic finite automata, but by determinacy, hardness also holds for universal automata, as they are dual.

Now, if a URA specification is realisable for some given k, then by Lemmas 1 and 3, $W^{\mathsf{QF}}_{S,k}$ is realisable by a finite-alphabet transducer M. Since $W^{\mathsf{QF}}_{S,k} \subseteq W^{\mathsf{F}}_{S,k}$, M also realises the specification $W^{\mathsf{F}}_{S,k}$. The mapping $\cdot_{synt}$ which turns a register transducer into a finite-alphabet transducer is bijective, and hence there exists a register transducer T such that $T_{synt} = M$. The proof of Lemma 1 exactly shows that T realises S, hence we are done. ◀

3.2 Register-bounded Synthesis over Data Domain $(\mathbb{N}, <, 0)$

We instantiate Theorem 4 for the data domain $(\mathbb{N}, <, 0)$. In [14], though there was no general notion of ω-regular approximability for data domains, it was implicitly used for $(\mathbb{N}, <, 0)$. The following fact follows from [14, Thm.8] after adapting to our notions.[2]

▶ **Fact 1.** For all R, $(\mathbb{N}, <, 0)$ has a witness QFEAS_R of ω-regular approximability expressible by a nondeterministic parity automaton with $exp(|R|)$ states and $poly(|R|)$ priorities, which can be constructed in time $exp(|R|)$.

A parity automaton can be translated to a nondeterministic Büchi automaton with a quadratic number of states, so we can instantiate Theorem 4 on domain $(\mathbb{N}, <, 0)$ and get:

▶ **Theorem 5.** *For a URA in $(\mathbb{N}, <, 0)$ with r registers, n states, and c priorities, k-register-bounded synthesis 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. It is* ExpTime-c *for fixed k and r.*

[2] Strictly speaking, their paper considers maximal tests only. However, using their deterministic automaton for QFEAS_R over action words with maximal tests, we can construct a *non*det. automaton recognising quasi-feasible action words with all tests, incl. partial ones. Our nondet. automaton, on reading a partial test, *guesses* its completion into a maximal test and simulates the original automaton on it.

4 Reducibility Between Data Domains

Theorem 5 relies on the study of feasibility of action words in $(\mathbb{N}, <, 0)$ of [14], which requires some effort. Such a study could in principle be generalised to domains such as $\mathbb{Z}$-tuples, as well as finite strings with the prefix relation, by leveraging the results of [10]. However, this would come at the price of a high level of technicality. We choose a different path, and introduce a notion of reducibility between domains, which allows us to reuse the study of $(\mathbb{N}, <, 0)$ and yields a compositional proof of the decidability of register-bounded synthesis for the quoted domains.

▶ **Definition.** A data domain $\mathcal{D}$ *reduces* to a data domain $\mathcal{D}'$ if for every finite set of registers R, there exists a finite set of registers R' and a rational relation[3] K between R-automata action words in $\mathcal{D}$ and R'-automata action words in $\mathcal{D}'$ that *preserves feasibility*, in the sense that for every R-action word $\bar{a} \in (\mathsf{Tst}_R^{\mathcal{D}} \mathsf{Asgn}_R)^\omega$: $\bar{a}$ is feasible in $\mathcal{D}$ iff there exists an R'-action word in $K(\bar{a}) \in (\mathsf{Tst}_{R'}^{\mathcal{D}'} \mathsf{Asgn}_{R'})^\omega$ feasible in $\mathcal{D}'$.[4]

▶ Remark. Reducibility is a transitive relation, since rational relations are closed under composition [3, Theorem 4.4], and feasibility preservation is transitive.

Since K is rational and preserves feasibility, for all R, $K^{-1}(\mathsf{QFEAS}_{R'})$ is a witness of regapproximability, where R' is as in the above definition (see the proof below for details), thus we get:

▶ **Lemma 6.** *If $\mathcal{D}$ reduces to $\mathcal{D}'$ and $\mathcal{D}'$ is regapprox, then $\mathcal{D}$ is regapprox.*

Proof. Let R be a fixed set of registers, and let R' be a set of registers satisfying the definition of reducibility. Let FEAS (respectively, FEAS') be the set of R-action words feasible in $\mathcal{D}$ (resp., feasible R'-action words in $\mathcal{D}'$).

Our goal is to define an ω-regular set QFEAS (for R) s.t. $\mathsf{QFEAS} \cap lasso \subseteq \mathsf{FEAS} \subseteq \mathsf{QFEAS}$. Since $\mathcal{D}'$ is regapprox, there is an ω-regular set QFEAS' (for R') s.t. $\mathsf{QFEAS}' \cap\ lasso \subseteq \mathsf{FEAS}' \subseteq \mathsf{QFEAS}'$. Define $\mathsf{QFEAS} = K^{-1}(\mathsf{QFEAS}')$; as the preimage of an ω-regular set by a rational relation, it is (effectively) ω-regular, thus satisfying one of the condition for $\mathcal{D}$ to be regapprox.

We now show that $\mathsf{FEAS} \subseteq \mathsf{QFEAS}$. Before proceeding, notice that $\mathsf{FEAS} = K^{-1}(\mathsf{FEAS}')$, since K preserves feasibility. 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 because $\mathsf{FEAS} = K^{-1}(\mathsf{FEAS}')$). We prove that $\mathsf{QFEAS} \cap lasso \subseteq K^{-1}(\mathsf{QFEAS}' \cap lasso)$, which entails the desired result. Pick an arbitrary $\bar{a} \in \mathsf{QFEAS} \cap lasso$. Since K is rational, $K(\bar{a})$ is ω-regular. Moreover, QFEAS' is ω-regular, which entails that $K(\bar{a}) \cap \mathsf{QFEAS}'$ is ω-regular as well. 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 $\bar{a} \in K^{-1}(\mathsf{QFEAS}' \cap lasso)$. ◀

As a direct consequence of Lemma 6 and Theorem 4, we get the following result:

---

3 Given two finite alphabets Σ and Γ, a relation $K \subseteq \Sigma^\omega \times \Gamma^\omega$ is rational if there exists an ω-regular language $L \subseteq (\Sigma \cup \Gamma)^\omega$ such that $K = \{(\mathrm{proj}_\Sigma(u), \mathrm{proj}_\Gamma(u)) \mid u \in L\}$. This is equivalent to saying that it can be computed by a nondeterministic asynchronous finite-state transducer over input Σ with output in Γ^*. See, e.g., [3, Section 3].

4 Note that we do not forbid the existence of unfeasible action words in the image.

▶ **Theorem 7.** *If a data domain $\mathcal{D}$ reduces to a regapprox data domain, then register-bounded synthesis is decidable for $\mathcal{D}$. Moreover, for any positive instance of the register-bounded synthesis problem over $\mathcal{D}$, one can effectively construct a register transducer realising the specification of that instance.*

4.1 Adding Labels to Data Values

As a first application, we show that one can equip data values with labels from a finite alphabet while preserving regapproximability. By Theorem 7, this yields decidability of register-bounded synthesis for such domains.

Formally, given a data domain $\mathcal{D} = (\mathbb{D}, P, C, c_0)$ and a finite alphabet Σ, we define the domain of Σ-labeled data values over $\mathcal{D}$ as $\Sigma \times \mathcal{D} = (\Sigma \times \mathbb{D}, P \cup \{\mathsf{lab}_\sigma \mid \sigma \in \Sigma\}, \Sigma \times C, (\sigma_0, c_0))$, where $\sigma_0 \in \Sigma$ is a fixed but arbitrary element of Σ and, for each $\sigma \in \Sigma$, $\mathsf{lab}_\sigma(\gamma, d)$ holds if and only if $\gamma = \sigma$.

▶ **Lemma 8.** *For all finite alphabet Σ and data domain $\mathcal{D}$, $\Sigma \times \mathcal{D}$ reduces to $\mathcal{D}$.*

Proof. Wlog we assume that the set of constants C is the singleton $\{c_0\}$ (modulo adding new predicates to P). Let $\Sigma = \{\sigma_0, \sigma_1, \dots, \sigma_n\}$, where σ_0 is such that (σ_0, c_0) is the initialiser of $\Sigma \times \mathcal{D}$. We first define an encoding at the level of data words. Let $\mu : \Sigma \to \mathbb{D}$ be an injective mapping such that $\mu(\sigma_0) = c_0$. A data word u over $\mathcal{D}$ is a μ-encoding of $v = (\sigma_{i_1}, d_1)(\sigma_{i_2}, d_2)\dots$ if it is equal to $\mu(\sigma_1)\dots\mu(\sigma_n)\mu(\sigma_{i_1})d_1\mu(\sigma_{i_2})d_2\dots$. The data word u is a *valid encoding* of v if it is a μ-encoding of v for some μ.

Now, the idea is to define a rational relation K from action words $\bar{a}$ over $\Sigma \times \mathcal{D}$ to actions words $\bar{b}$ over $\mathcal{D}$ such that $\bar{a}$ is feasible by some u iff there exists $\bar{b}$ such that $(\bar{a}, \bar{b}) \in K$ and $\bar{b}$ is feasible by a valid encoding of u. Let R be a set of registers and assume $\bar{a}$ is built over R. Let $R' = \{r_\sigma \mid \sigma \in \Sigma\} \uplus R$. Then, any $\bar{b}$ such that $(\bar{a}, \bar{b}) \in K$ should ensure that the n first data values are distinct and store them in $r_{\sigma_1}, \dots, r_{\sigma_n}$ respectively. So, we require that $\bar{b}$ is of the form $\bar{b} = b_\Sigma \cdot b_{\bar{a}}$ where $b_\Sigma = (\mathsf{tst}_{\sigma_1}, \downarrow r_{\sigma_1})\dots(\mathsf{tst}_{\sigma_n}, \downarrow r_{\sigma_n})$ such that for all $1 \le i \le n$, $\mathsf{tst}_i = \bigwedge_{1 \le j \le i} \star \neq r_{\sigma_j}$. The second part $b_{\bar{a}}$ is an encoding of the tests and assignments of $\bar{a} = (\mathsf{tst}_0, \mathsf{asgn}_0)(\mathsf{tst}_1, \mathsf{asgn}_1)\dots$. It is of the form $b_{\bar{a}} = (\mathsf{tst}_0^{lab}, \varnothing)(\mathsf{tst}_0^{data}, \mathsf{asgn}_0)(\mathsf{tst}_1^{lab}, \varnothing)(\mathsf{tst}_1^{data}, \mathsf{asgn}_1)\dots$, where for all $i \ge 0$:

- for every predicate $p \in P$ of arity n, for every $x_1, \dots, x_n \in R \cup \{\star\}$: if $(\neg)p(x_1, \dots, x_n) \in \mathsf{tst}_i$, then $(\neg)p(x_1, \dots, x_n) \in \mathsf{tst}_i^{data}$, and
- for all $\sigma \in \Sigma$ and $x \in R \cup \{\star\}$: $\mathsf{lab}_\sigma(x) \in \mathsf{tst}_i^{lab}$ iff $(r_\sigma = x) \in \mathsf{tst}_i$.

Correctness follows from the construction; see Appendix B for details. ◀

The latter result combined with Theorem 7 yields:

▶ **Corollary 9.** *Let $\mathcal{D}$ be an regapprox data domain and Σ be a finite alphabet, then register-bounded synthesis is decidable for $\Sigma \times \mathcal{D}$.*

4.2 Quantifier-Free Interpretations

When the relation between valuations over $\mathcal{D}$ and over $\mathcal{D}'$ is local, it is more convenient to operate directly at the level of tests. To that end, we define a notion of quantifier-free interpretation (see [13, Section 12.3.6] for a presentation of the notion in the context of data words), that allows us to encode elements of $\mathcal{D}$ as tuples of elements of $\mathcal{D}'$.

A *quantifier-free interpretation* (or *interpretation* for short) of dimension $l \ge 1$ with signature (P, C) over a data domain $\mathcal{D}' = (\mathbb{D}', P', C')$ is given by quantifier-free formulas over signature (P', C'). The formula $\phi_{\text{domain}}(x_1, \dots, x_l)$ defines the domain $\mathbb{D} =$

$\{(d_1,\dots,d_l) \mid \mathcal{D}' \models \phi_{\text{domain}}(d_1,\dots,d_l)\}$. Then, for each constant symbol $c \in C$, the formula $\phi_c(x_1,\dots,x_l)$ defines the encodings[5] of c as the tuples $(d_1^c,\dots,d_l^c) \in \mathbb{D}$ that satisfy ϕ_c, i.e. such that $\mathcal{D}' \models \phi_c(d_1^c,\dots,d_l^c)$. Finally, for each predicate $p \in P$ of arity a (including $=$), the formula $\phi_p(x_1^1,\dots,x_l^1,\dots,x_1^a,\dots,x_l^a)$ defines the predicate $p^{\mathcal{D}} = \{(d_1^1,\dots,d_l^1,\dots,d_1^a,\dots,d_l^a) \mid \mathcal{D}' \models \phi_R(d_1^1,\dots,d_l^1,\dots,d_1^a,\dots,d_l^a)\}$.

▶ **Lemma 10.** $(\mathbb{Z},<,0)$ *can be defined as a 2-dimensional interpretation of* $(\mathbb{N},<,0)$.

Proof. The encoding consists of two copies of $\mathbb{N}$, one for positive and one for negative integers, whose order is reversed. Formally, $\phi_{domain}(x_1,x_2) \coloneqq x_1 = 0 \vee x_2 = 0$. Then, $\phi_0(x_1,x_2) \coloneqq x_1 = 0 \wedge x_2 = 0$; $\phi_=((x_1,x_2),(y_1,y_2)) \coloneqq x_1 = y_1 \wedge x_2 = y_2$ and $\phi_<((x_1,x_2),(y_1,y_2)) \coloneqq (x_2 = y_2 = 0 \wedge x_1 < y_1) \vee (x_1 = y_1 = 0 \wedge x_2 > y_2) \vee (x_1 = 0 \wedge y_1 > 0)$. Then, $(\mathbb{Z},<,0)$ is isomorphic to this structure, through the bijection $n \geq 0 \mapsto (n,0)$ and $n < 0 \mapsto (0,-n)$. ◀

More generally, d-uples of integers can be easily encoded. In the following, we fix $d \geq 1$. For $(n_1,...,n_d),(m_1,...,m_d) \in \mathbb{Z}^d$, define $(n_1,...,n_d) <^d (m_1,...,m_d)$ iff for all $i \in \{1,...,d\}$, $n_i \leq m_i$ and $n_j < m_j$ for some $j \in \{1,\dots,d\}$; it is a partial order on $\mathbb{Z}^d$. The predicate $=^d$ is defined as expected.

▶ **Lemma 11.** $(\mathbb{Z}^d,=^d,<^d,0^d)$ *can be defined as a d-dimensional interpretation of* $(\mathbb{Z},<,0)$.

Proof. Any tuple belongs to the domain, so we let $\phi_{\text{domain}} \coloneqq \top$. Then, $\phi_0(x_1,\dots,x_d) \coloneqq \bigwedge_{1\leq i\leq d} x_i = 0$, $\phi_=((x_1,\dots,x_d),(y_1,\dots,y_d)) \coloneqq \bigwedge_{1\leq i\leq d} x_i = y_i$, and similarly for $\phi_<$. ◀

The following theorem allows us to lift our results to the two domains above:

▶ **Theorem 12.** *If* $\mathcal{D}$ *is a quantifier-free interpretation over* $\mathcal{D}'$*, then* $\mathcal{D}$ *reduces to* $\mathcal{D}'$.

Proof (Sketch). We outline the proof, and refer to Appendix B.2 for details. Let $\mathcal{D}' = (\mathbb{D}',P',C')$ be a data domain, and $\mathcal{D}$ be an interpretation over $\mathcal{D}'$ of dimension $l \geq 1$ with signature (P,C). The main idea is, given a set of registers R, to consider l copies of this set, meant to store each dimension of the interpretation. We also add l copies of C to store the encoding of constants, and, since tests are conducted before assignment, l registers to store each component of the input tuple. Overall, an action word $\bar{a}$ over R is sent to one over $(R \cup C \cup \{d\}) \times \{1,\dots,l\}$, where d is a fresh register variable. Then we construct the sought relation K as follows: first, it prefixes its image with a sequence of actions that store the encoding of constants in the corresponding registers, check that they indeed satisfy their respective ϕ_c, and ensure that all registers in $R \times \{1,\dots,l\}$ are initialised with the encoding of c_0. Note that the formulas are not necessarily conjuncts, so we put them in disjunctive normal form and consider all tests that are conjuncts of the DNF. Then, each action is processed separately: an action $(\mathsf{tst},\mathsf{asgn})$ of $\bar{a}$ is associated with a sequence of $2l+1$ actions that consist in reading each component of the input data value $\star$, store it in the corresponding copy of d, check that $\star$ indeed belongs to the domain (ϕ_{domain}), and that it satisfies tst (using the $(\phi_p)_{p\in P}$ to encode the predicates). Again, this implies converting the formulas in DNF, so a given action is in general associated with multiple ones. Since K consists in adding a prefix and then processing each action separately, it is rational. Moreover, it preserves feasibility; more precisely for any action word $\bar{a}$, each of its corresponding data word can be associated with its encoding in $K(\bar{a})$. ◀

By Theorems 5, 12 and 7, as well as Lemma 10, we get:

[5] Note that we do not assume the encoding to be unique.

▶ **Corollary 13.** *Register-bounded synthesis is decidable for* $(\mathbb{Z}, <, 0)$.

Then, since $(\mathbb{Z}, <, 0)$ reduces to $(\mathbb{N}, <, 0)$, and reducibility is transitive, we get, by Lemma 11 and Theorems 12 and 7:

▶ **Corollary 14.** *Register-bounded synthesis is decidable for* $(\mathbb{Z}^d, =^d, <^d, 0^d)$.

▶ Remark. One can similarly show that $\mathbb{N}^d$ reduces to $\mathbb{N}$. More generally, the above method allows one to lift decidability of register-bounded synthesis to tuples of data values where predicates are applied component-wise. Besides, note that $\mathbb{N}^d$ also reduces to $\mathbb{Z}^d$, by restricting $\mathbb{Z}^d$ to nonnegative values.

4.3 Finite Strings with the Prefix Relation

In this section, we show that synthesis is decidable over the data domain $(\Sigma^*, =, \prec, \epsilon)$, where Σ is a finite alphabet and $\prec$ denotes the prefix relation, leveraging a result of [10] that encodes prefix constraints as integer ones. This still requires some work, as we cannot use the notion of interpretation: a string valuation is encoded as an integer valuation with a *quadratic* number of registers. In the sequel, Σ is a fixed finite set of size $l \geq 2$.

First, $(\Sigma^*, =, \prec, \epsilon)$ reduces to the richer domain $(\Sigma^*, =, \mathit{clen}_=, \mathit{clen}_<, \epsilon)$, where, given $u, v \in \Sigma^*$, $\mathit{clen}(u, v)$ denotes the length of the longest common prefix of u and v, and, for $\lhd \in \{<, =\}$, $\mathit{clen}_\lhd(u, v, u', v')$ holds whenever $\mathit{clen}(u, v) \lhd \mathit{clen}(u', v')$. The reduction is direct, and follows the same lines as [10, Lemma 3]: $u \prec v$ is encoded as $(\mathit{clen}(u, u) = \mathit{clen}(u, v)) \wedge (\mathit{clen}(u, u) < \mathit{clen}(v, v))$, and K is a morphism on tests and the identity over assignments.

▶ **Lemma 15.** $(\Sigma^*, =, \prec, \epsilon)$ *reduces to* $(\Sigma^*, =, \mathit{clen}_=, \mathit{clen}_<, \epsilon)$.

Note also that satisfiability of tests over both domains is decidable, and NP-complete [10, Lemma 7]. It now remains to show that $(\Sigma^*, =, \mathit{clen}_=, \mathit{clen}_<, \epsilon)$ reduces to $(\mathbb{N}, =, <, 0)$. The proof draws on ideas similar to that of [10, Lemmas 8,9], which mainly relies on [10, Lemmas 5,6]. Here, it remains to lift them to our synthesis framework, and ensure that feasibility is preserved despite the dependencies induced by registers.

▶ **Lemma 16.** $(\Sigma^*, =, \mathit{clen}_=, \mathit{clen}_<, \epsilon)$ *reduces to* $(\mathbb{N}, =, <, 0)$.

Proof. We describe the main ideas of the proof; a full proof can be found in Appendix B.3. From [10, Lemma 5,6], we know that a string valuation is characterised by the length of the longest common prefixes of all its pairs of values, when prefix constraints are concerned. This allows to encode Σ^* in $\mathbb{N}$: given a set R of registers, we introduce a register $\pi_{r,s}$ for each $(r, s) \in R' = (R \cup \{\mathsf{x}\})^2$, where x is an additional register name that denotes the input data value $\star$ in Σ^*. Along the execution, a register $\pi_{r,s}$ is meant to contain $\mathit{clen}(\nu(r), \nu(s))$. Note that in particular, $\pi_{r,r}$ contains the length of the word stored in r. At each step, we read a sequence of $|R|$ integers that each corresponds to the value of $\mathit{clen}(\star, r)$ for some $r \in R$, that we store in the corresponding register $\pi_{\star,r}$. We then check that they satisfy the *clen* constraints, as well as the properties of [10, Proposition 2]. The latter consist in logical formulas that can be encoded as tests in $(\mathbb{N}, =, <, 0)$, as they only use $=$ and $<$.

Using [10, Lemma 6], from a sequence of integer valuations (called *counter valuations* in [10]) that satisfy those properties, we can reconstruct a sequence of string valuations. As the integer valuations additionally satisfy the *clen* constraints, so does the string valuations. Thus, if an image R'-action word is feasible, the original action word is feasible. The converse direction is easier: given a sequence $\nu_0\nu_1\ldots$ of string valuations that is compatible with the R-action word, at step i one fills each $\pi_{r,s}$ with $\mathit{clen}(\nu_i(r), \nu_i(s))$. ◀

By Theorems 5 and 7, we get:

▶ **Corollary 17.** *Register-bounded synthesis is decidable for* $(\Sigma^*, =, \prec, \epsilon)$.

▶ Remark 18 (Complexity analysis). Note that the data domains in Corollaries 13, 14 and 17 all reduce to $(\mathbb{N}, <, 0)$ (all via some rational relations K depending on a set of registers R). The time complexities of those corollaries depend on the complexities of constructing, given a set of registers R, a nondeterministic Büchi automaton recognising $K^{-1}(\mathsf{QFEAS}_R^{(\mathbb{N},<,0)})$ for all the rational relations K defined in the proofs of those corollaries. It can be seen from those proofs that for any such rational relation K, it is possible to construct a nondeterministic Büchi transducer A_K with polynomially many states in $|R|$ recognising K. By taking the synchronized product of A_K with a nondeterministic automaton recognising $\mathsf{QFEAS}_R^{(\mathbb{N},<,0)}$, say of size n_{qf}, and by projecting it on its inputs, one obtains a nondeterministic Büchi automaton recognising $K^{-1}(\mathsf{QFEAS}_R^{(\mathbb{N},<,0)})$. It can be computed in time $poly(n_{qf})$. By Fact 1 and Theorem 4, one gets that the time complexities of k-register-bounded synthesis for data domains $(\mathbb{Z}, =, <, 0)$, $(\mathbb{Z}^d, =^d, <^d, 0^d)$ (for a fixed d) and $(\Sigma^*, =, \prec, \epsilon)$ is doubly exponential in k and r the number of registers of the specification, and singly exponential in the number of states of the URA and its number of priorities.

5 Conclusion

We have shown that register-bounded synthesis from specifications expressed by universal register-automata over $(\mathbb{N}, <, 0)$ is decidable within the same time complexity class as the case of URA over $(\mathbb{N}, =)$, completing the picture on synthesis from register automata over $(\mathbb{N}, =)$ and $(\mathbb{N}, <, 0)$: (unbounded) synthesis is undecidable for nondeterministic register automata [15], decidable for deterministic register automata over $(\mathbb{N}, =)$ [15] and over $(\mathbb{N}, <)$ [14], and register-bounded synthesis is decidable for URA over $(\mathbb{N}, =)$ [24, 15, 25] and $(\mathbb{N}, <, 0)$ (this paper), and undecidable for nondeterministic register automata [15]. We also get decidability for the data domains of integers, of tuples of integers and of finite words with the prefix relation, by reducing them to $(\mathbb{N}, <, 0)$. A simple complexity analysis (Remark 18) yields a doubly exponential decision procedure for register-bounded synthesis over these domains. Systematising this complexity analysis calls for a notion of polynomial reduction between data domains, that we leave for future work.

There are other 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 [26, 31, 17, 4]. We plan on investigating extensions of these works to the register setting. In particular, our synthesis algorithm first reduces the problem to a synthesis problem over a *finite* alphabet with a specification given by a universal co-Büchi automaton. The latter problem is classically solved by reduction to a parity game obtained by determinising the universal co-Büchi automaton, e.g. by using Safra's determinization procedure. It is an interesting question whether Safraless procedures from [26, 31, 17] 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, as the nice correspondences between automata and logics for word languages over finite alphabets do not carry over to data words. Nevertheless, URA encompass Constraint LTL [32], and we believe their expressive power could allow one to target other temporal-like logics with data.

References

1 Shaull Almagor and Orna Kupferman. Good-enough synthesis. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020. doi:10.1007/978-3-030-53291-8_28.

2 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.

3 Jean Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher : Informatik*. Teubner, 1979. URL: https://www.worldcat.org/oclc/06364613.

4 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking*, pages 921–962. Springer, 2018.

5 M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 285–296, 2006.

6 Mikołaj Bojańczyk. *Slightly infinite sets*. Mikołaj Bojańczyk, 2019. URL: https://www.mimuw.edu.pl/~bojan/paper/atom-book.

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

8 Alex Bystrov, David John Kinniment, and Alexandre Yakovlev. Priority arbiters. In *Proceedings Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)(Cat. No. PR00586)*, pages 128–137. IEEE, 2000.

9 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.

10 Stéphane Demri and Morgan Deters. Temporal logics on strings with prefix relation. *Journal of Logic and Computation*, 26(3):989–1017, 2016.

11 Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint ltl. *Information and Computation*, 205(3):380–415, 2007. URL: https://www.sciencedirect.com/science/article/pii/S0890540106001076, doi:https://doi.org/10.1016/j.ic.2006.09.006.

12 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.

13 Léo Exibard. *Automatic Synthesis of Systems with Data*. PhD Thesis, Aix-Marseille Université (AMU); Université libre de Bruxelles (ULB), September 2021. URL: http://www.icetcs.ru.is/leoe/files/Exibard_ASSD_SASD.pdf.

14 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.

15 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of Data Word Transducers. *Logical Methods in Computer Science*, Volume 17, Issue 1, March 2021. URL: https://lmcs.episciences.org/7279, doi:10.23638/LMCS-17(1:22)2021.

16 Rachel Faran and Orna Kupferman. On synthesis of specifications with arithmetic. In Alexander Chatzigeorgiou, Riccardo Dondi, Herodotos Herodotou, Christos Kapoutsis, Yannis Manolopoulos, George A. Papadopoulos, and Florian Sikora, editors, *SOFSEM 2020: Theory and Practice of Computer Science*, pages 161–173, Cham, 2020. Springer International Publishing.

17 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.

18 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 125:1–125:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.ICALP.2016.125`.

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

20 O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *Proc. 4th Int. Conf. on Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, pages 561–572. Springer, 2010.

21 Ranjit Jhala, Andreas Podelski, and Andrey Rybalchenko. Predicate abstraction for program verification. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 447–491. Springer, 2018. `doi:10.1007/978-3-319-10575-8\_15`.

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

23 M. Kaminski and D. Zeitlin. Extending finite-memory automata with non-deterministic reassignment. In *AFL*, pages 195–207, 2008.

24 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.

25 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`.

26 O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

27 Benedikt Maderbacher and Roderick Bloem. Reactive synthesis modulo theories using abstraction refinement, 2021. `arXiv:2108.00090`.

28 F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In *26th Int. Symp. on Mathematical Foundations of Computer Science*, pages 560–572. Springer-Verlag, 2001.

29 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.

30 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.

31 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.

32 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.

33 N. Tzevelekos. Fresh-register automata. In *Proc. 38th ACM Symp. on Principles of Programming Languages*, pages 295–306, New York, NY, USA, 2011. ACM.

A Detailed Proofs of Section 3

A.1 Proof of Lemma 2

Proof. We prove the second claim, implying the first one. Let $R = R_S \uplus R_k$. Let QF be a nondet. Büchi automaton recognising QFEAS_R with the sizes mentioned by the lemma. Note that $|\mathsf{Tst}_R|$ is $exp(r,k)$ and $|\mathsf{Asgn}_R| = 2^{r+k}$ is $exp(r,k)$. Recall that the specification automaton S works on words that interleave input and output data of transducers. We make this explicit by assuming the states of S are partitioned into input and output states for reading input and output data. The automaton S can be transformed into this form at the cost of doubling the number of states and in time $poly(n, exp(r))$. The transitions from input states are called input transitions, from output states – on output transitions. The input alphabet is that of the possible letters on input transitions, output alphabet – on output transitions. We now construct the sought universal automaton for $W^{\mathsf{QF}}_{S,k}$.

1. Construct the nondet. Büchi automaton QF' from QF as follows: replace every input transition labelled $(\mathsf{tst}, \mathsf{asgn}) \in \mathsf{Tst}_R \times \mathsf{Asgn}_R$ by transitions labelled $(\mathsf{tst}, \mathsf{asgn}, r^k)$ for every $r^k \in R_k$. Hence, the projection of $L(QF')$ on $\mathsf{Tst}_R \times \mathsf{Asgn}_R$ equals QFEAS_R. Thus, the automaton QF' can be constructed in time $poly(N, exp(r,k))$. We assume the transition relation of QF' is complete (to be able to complement it via dualising). If it is not complete, we can make it such in time $poly(N, exp(r,k))$.

2. Let S_{synt} be the syntactic view of S; it is a universal co-Büchi automaton with input and output finite alphabets $\mathsf{Tst}_S \times \mathsf{Asgn}_S$. We assume that the transition relation of S_{synt} is complete, which can be enforced in time $poly(n, exp(r))$. Let $\widetilde{S}_{synt}$ be the nondet. parity automaton dual to S_{synt}, thus $L(\widetilde{S}_{synt}) = \overline{S_{synt}}$. We construct $\widetilde{S}'_{synt}$ from $\widetilde{S}_{synt}$ as follows: extend the input and output alphabets of $\widetilde{S}_{synt}$ from $\mathsf{Tst}_S \times \mathsf{Asgn}_S$ to $\mathsf{Tst}_R \times \mathsf{Asgn}_R$ and to $\mathsf{Tst}_R \times \mathsf{Asgn}_R \times R_k$, while preserving the original literals, formally as follows. Every input transition of $\widetilde{S}_{synt}$ labelled $(\mathsf{tst}^{\mathsf{i}}, \mathsf{asgn}^{\mathsf{i}}) \in \mathsf{Tst}_S \times \mathsf{Asgn}_S$ is replaced by the output transitions labelled $(\mathsf{tst}^{\mathsf{i}\prime}, \mathsf{asgn}^{\mathsf{i}\prime}) \in \mathsf{Tst}_R \times \mathsf{Asgn}_R$ s.t. $\mathsf{tst}^{\mathsf{i}} \subseteq \mathsf{tst}^{\mathsf{i}\prime}$ and $\mathsf{asgn}^{\mathsf{i}} \subseteq \mathsf{asgn}^{\mathsf{i}\prime}$. Similarly, every output transition labelled $(\mathsf{tst}^{\mathsf{o}}, \mathsf{asgn}^{\mathsf{o}}) \in \mathsf{Tst}_S \times \mathsf{Asgn}_S$ is replaced by the output transitions labelled $(\mathsf{tst}^{\mathsf{o}\prime}, \mathsf{asgn}^{\mathsf{o}\prime}, r) \in \mathsf{Tst}_R \times \mathsf{Asgn}_R \times R_k$ s.t. $\mathsf{tst}^{\mathsf{o}} \subseteq \mathsf{tst}^{\mathsf{o}\prime}$ and $\mathsf{asgn}^{\mathsf{o}} \subseteq \mathsf{asgn}^{\mathsf{o}\prime}$. This can be done in time $poly(n, exp(r,k))$.

3. We now construct $QF' \wedge \widetilde{S}'_{synt}$. First, translate the nondet. parity automaton $\widetilde{S}'_{synt}$ into a nondet. Büchi automaton with $O(nc)$ states. Then, build the product of all automata and get a nondet. Büchi automaton with $O(Nnc)$ states. This can be done in time $poly(N, n, exp(r,k))$. Notice that every word accepted by $QF' \wedge \widetilde{S}'_{synt}$ is a product of some transducer action word $\bar{a}_k$ and automaton action word $\bar{a}_S$ (equipped with labels r_k on output actions to keep track of the output register) such that $\bar{a}_S$ is rejected by S_{synt}.

4. Project $QF' \wedge \widetilde{S}'_{synt}$ into the input alphabet $\mathsf{Tst}_k \times \mathsf{Asgn}_k$ and output alphabet R_k. This does not affect the number of states and can be done in time $poly(N, n, exp(r,k))$.

5. Shift the component Asgn_k from the input alphabet to the output alphabet, which multiplies the number of states by $|\mathsf{Asgn}_k| = 2^k$, so the number of states becomes $O(2^k Nnc)$. This can be done in time $poly(N, n, exp(r,k))$. Call the result A'. Note that $L(A') = \overline{W^{\mathsf{QF}}_{S,k}}$.

6. Finally, we treat the nondet. Büchi automaton A' as universal co-Büchi, and this is a sought universal co-Büchi automaton with $O(2^k Nnc)$ many states.

The correctness should be clear as the construction follows the definition of $W^{\mathsf{QF}}_{S,k}$. ◀

B Detailed Proofs of Section 4

B.1 Proof of Lemma 8

Proof. It remains to show that the construction is correct. First, K is rational: a finite transducer just has to first output $\overline{b_\Sigma}$ (which is independent from $\bar{a}$) and then, whenever it reads $(\mathsf{tst}_i, \mathsf{asgn}_i)$ in $\bar{a}$, it outputs any $(\mathsf{tst}_i^{lab}, \varnothing)(\mathsf{tst}_i^{data}, \mathsf{asgn}_i)$ which satisfies the two latter conditions (there are only finitely many). Let us show that K preserves feasibility. Suppose $\bar{a}$ is feasible by $u = (\sigma_{i_1}, d_1)(\sigma_{i_2}, d_2)\dots$, then since $\mathcal{D}$ is assumed to be infinite, there exists an injective mapping $\mu : \Sigma \to \mathcal{D}$ such that $\mu(\sigma_0) = c_0$. Consider the word $v = \mu(\sigma_1)\dots\mu(\sigma_n)\mu(\sigma_{i_1})d_1\mu(\sigma_{i_2})d_2\dots$. It can be checked that there exists a unique action word $\bar{b}$ feasible by v of the form $\overline{b_\Sigma}.\overline{b_{\bar{a}}}$ and by construction of K we have $(\bar{a}, \bar{b}) \in K$.

Conversely, if $\bar{b}$ is feasible by some $v = e_1e_2\dots e_n d'_1 d_1 d'_2 d_2 \dots$ such that $(\bar{a}, \bar{b}) \in K$, then, by letting $e_0 = c_0$ we have, by construction of K, that $e_i \neq e_j$ for all $0 \leq i < j \leq n$, and for all $i \geq 1$, there exists $0 \leq j \leq n$ such that $d'_i = e_j$. By letting $\mu : \sigma_i \mapsto e_i$ for all $0 \leq i \leq n$, we get that v is a μ-encoding of $(\mu^{-1}(d'_1), d_1)(\mu^{-1}(d'_2), d_2)\dots$ which by construction of K is a witness of feasibility of $\bar{a}$. ◀

B.2 Proof of Theorem 12

Proof. Let $\mathcal{D}$ and $\mathcal{D}'$ be two data domains such that $\mathcal{D}$ is a quantifier-free interpretation of $\mathcal{D}'$ of dimension $l \geq 1$, with predicates P and constants $C = \{c_0, \dots, c_m\}$, given by formulas ϕ_{domain}, $\{\phi_c \mid c \in C\}$ and $\{\phi_p \mid p \in P\}$; without loss of generality we assume that $\phi_c \Rightarrow \phi_{domain}$ for all $c \in C$. Note that the formulas are not necessarily conjuncts (there can be disjunctions and negations), so a formula might induce multiple tests.

Let R be a set of registers; we assume without loss of generality that R and C are disjoint and let $d \notin (R \cup C)$ be a fresh register. We define $R' = (R \cup C \cup \{d\}) \times \{1, \dots, l\}$ and write r^i for (r, i). We need to define a feasibility-preserving rational relation K between action words over R and R'. The images of an action word $\bar{a}$ through K consists in a prefix π followed by $\kappa(\bar{a})$, where κ is defined at the level of action words and extended homomorphically.

Prefixes are of the form $\pi = (\top, c_0^1 \cup R \times \{1\})\dots(\top, c_0^l \cup R \times \{l\})(\phi'_{c_0}(c_0^1, \dots, c_0^l), \varnothing) \cdot (\top, c_1^1)\dots(\top, c_1^l)(\phi'_{c_1}(c_1^1, \dots, c_1^l), \varnothing)\dots(\top, c_m^1)\dots(\top, c_m^l)(\phi'_{c_m}(c_m^1, \dots, c_m^l), \varnothing)$, where the ϕ'_c are tests (i.e. conjuncts) that are implied by ϕ_c. It consists in reading the encoding of c_0 and storing it in registers $c_0^1, \dots, c_0^l$ as well as in registers of $R \times \{1, \dots, l\}$, and then reading the encoding of the other constants, storing them in the corresponding registers, and checking that the encodings are correct along the run.

We now define κ at the level of actions over R. The idea is to first read the l elements of the tuple encoding the input data $\star$, store it in the l copies of register d, check that the read value indeed belongs to the domain and satisfies the test, and store it in the registers of asgn by reading it a second time. Formally, an action $(\mathsf{tst}, \mathsf{asgn}) \in \mathsf{Tst}_R \times \mathsf{Asgn}_R$ is associated with the sequence of $2l + 1$ actions $(\top, d^1)\dots(\top, d^l) \cdot (\phi'_{\text{domain}}(d^1, \dots, d^l) \wedge \tau(\mathsf{tst})) \cdot (\star = d^1, \mathsf{asgn} \times \{1\})\dots(\star = d^l, \mathsf{asgn} \times \{l\})$, where $\phi'_{\text{domain}}(d^1, \dots, d^l)$ is a test that is a consequence of $\phi_{\text{domain}}(d^1, \dots, d^l)$, and where τ is defined on atomic tests as $\tau(p(x_1, \dots, x_a)) = \phi_p(x_1^1, \dots, x_1^l, \dots, x_a^1, \dots, x_a^l)$ for all predicates $p \in P$ of arity $a \in \mathbb{N}$, replacing x with d if $x = \star$ and applied pointwise to each conjunct.

Now, if $\bar{a}$ is feasible in R by some sequence $(d_0^1, \dots, d_0^l)(d_1^1, \dots, d_1^l)\dots$, then $K(\bar{a})$ is feasible by the sequence $c_0^1 \dots c_0^l e_0 \dots c_m^1 c_m^l e_m d_0^1 \dots d_0^l f_0 d_0^1 \dots d_0^l d_1^1 \dots d_1^l f_1 d_1^1 \dots d_1^l \dots$, where $e_0, f_0, e_1, f_1, \dots$ are arbitrary elements of $\mathbb{D}$. Conversely, assume that $K(\bar{a})$ is feasible by some sequence $c_0^1 \dots c_0^l e_0 \dots c_m^1 c_m^l e_m d_0^1 \dots d_0^l f_0 {d'}_0^1 \dots {d'}_0^l d_1^1 \dots d_1^l f_1 {d'}_1^1 \dots {d'}_1^l \dots$. Since formulas are checked through the tests, the encoding of each constant $c \in C$ satisfies $\phi_c(c^1, \dots, c^l)$, and after reading π registers in $R \times \{1, \dots, l\}$ contain the encoding of c_0. Moreover, we necessarily have $d_i^j = d'^j_i$. We also know that each $(d_i^1, \dots, d_i^l)$ satisfies ϕ_{domain}, and that the tests are satisfied by construction of τ, so it means that $\bar{a}$ is feasible by $(\mathscr{d}_0^1, \dots, \mathscr{d}_0^l)(\mathscr{d}_1^1, \dots, \mathscr{d}_1^l) \dots$, $K(\bar{a})$. Moreover, K is rational, as it it defined homomorphically, so $\mathcal{D}$ indeed reduces to $\mathcal{D}'$. ◀

B.3 Proof of Lemma 16

Proof. Let $R = \{r_1, \dots, r_k\}$ be a finite set of registers, and let $\bar{a}$ be an action word over R. We let R' be the set of unordered pairs over $R \cup \{\mathsf{x}\}$, where x is an additional variable name that aims at denoting the input word at each step. For readability, a register $\{r, s\} \in R'$ is denoted $\pi_{r,s}$. Note that we only need unordered pairs, since *clen* is a symmetric function. For clarity, in the following, we import the terminology of [10]: a register valuation $\nu : R \to \Sigma^*$ is called a *string valuation*, while a register valuation $\nu' : R' \to \mathbb{N}$ is a *counter valuation*. Here, we cannot use the notion of quantifier-free interpretation, since the encoding of a valuation over R necessitates a number of registers that is quadratic in R, and interpretations only allow linear encodings.

Each action $(\mathsf{tst}, \mathsf{asgn})$ of $\bar{a}$ translates to a sequence of actions as follows:

- First, we read the values of $\mathit{clen}(\star, r)$ for each $r \in R \cup \{\mathsf{x}\}$, and store them. This corresponds to a sequence $(\mathsf{tst}_1, \{\pi_{r_1,\mathsf{x}}\}) \dots (\mathsf{tst}_k, \{\pi_{r_k,\mathsf{x}}\})(\mathsf{tst}_\star, \{\pi_{\mathsf{x},\mathsf{x}}\})$; at this point we put no constraints on the incoming data values so $\mathsf{tst}_1, \dots, \mathsf{tst}_k, \mathsf{tst}_\star$ can be any tests over $(\mathbb{N}, <, 0)$. Note that the last value corresponds to the length of the input data value (which is a word over Σ).
- Then, $\bar{a}'$ checks that the values that have been read indeed yield a string-compatible counter valuation, in the sense of [10, Proposition 2 and Section 3], i.e. a counter valuation from which one can reconstruct a string valuation. Any action $(\mathsf{tst}_{\mathit{clen}}, \varnothing)$ such that $\mathsf{tst}_{\mathit{clen}}$ satisfies the following enforces that it is indeed the case:
  - If $\mathit{clen}_=(r, s, r', s') \in \mathsf{tst}$, then $\mathsf{tst}_{\mathit{clen}}$ should contain $\pi_{r,s} = \pi_{r',s'}$
  - If $\mathit{clen}_<(r, s, r', s') \in \mathsf{tst}$, then $\mathsf{tst}_{\mathit{clen}}$ should contain $\pi_{r,s} < \pi_{r',s'}$
  - If $r = s \in \mathsf{tst}$, then $\mathsf{tst}_{\mathit{clen}}$ should contain $\pi_{r,r} = \pi_{r,s} = \pi_{s,s}$
  - If $r = \epsilon \in \mathsf{tst}$, then $\pi_{r,r} = 0 \in \mathsf{tst}_{\mathit{clen}}$
  - If $\neg\mathit{clen}_=(r, s, r', s') \in \mathsf{tst}$, then $\mathsf{tst}_{\mathit{clen}}$ should contain $\pi_{r,s} < \pi_{r',s'}$ or $\pi_{r,s} > \pi_{r',s'}$
  - If $\neg\mathit{clen}_<(r, s, r', s') \in \mathsf{tst}$, then $\mathsf{tst}_{\mathit{clen}}$ should contain $\pi_{r,s} = \pi_{r',s'}$ or $\pi_{r,s} > \pi_{r',s'}$
  - If $\neg(r = s) \in \mathsf{tst}$, then $\mathsf{tst}_{\mathit{clen}}$ should contain $\pi_{r,r} < \pi_{r,s}$ (r is a strict prefix of s) or, symmetrically, $\pi_{s,s} < \pi_{r,s}$, or $\pi_{r,s} < \pi_{r,r}$ or $\pi_{r,s} < \pi_{s,s}$ (they mismatch at some point)
  - If $\neg(r = \epsilon) \in \mathsf{tst}$, then $\pi_{r,r} > 0 \in \mathsf{tst}_{\mathit{clen}}$ Additionally, we require that $\mathsf{tst}_{\mathit{clen}}$ implies $\psi_I \wedge \psi_{II} \wedge \psi_{III}$, where $\psi_I, \psi_{II}, \psi_{III}$ are defined in [10, Section 3.2] as the syntactical counterparts of the formulas ϕ_I, ϕ_{II} and ϕ_{III} of [10, Proposition 2], and characterise counter valuations that are string-compatible [10, Lemmas 5,6]. We recall them here, so that the reader can observe that they only depend on the predicates of $(\mathbb{N}, <, 0)$.
    - $\psi_I = \bigwedge_{r,r' \in R \cup \{\mathsf{x}\}} (\pi_{r,r} \geq \pi_{r,r'})$
    - $$\psi_{II} = \bigwedge_{r^0,\dots,r^l \in R \cup \{\mathsf{x}\}} \left(\left(\bigwedge_{0 \leq i \leq l} (\pi_{r^0,r^1} < \pi_{r^i,r^i}) \right) \wedge \pi_{r^0,r^1} = \dots = \pi_{r^0,r^l} \right) \Rightarrow \left(\bigwedge_{1 \leq i < j \leq l} (\pi_{r^0,r^1} < \pi_{r^i,r^j}) \right)$$
    - $\psi_{III} = \bigwedge_{r,r',r'' \in R \cup \{\mathsf{x}\}} (\pi_{r,r'} < \pi_{r',r''}) \Rightarrow (\pi_{r,r'} = \pi_{r,r''})$

Note that all register variables range over $R \cup \{\mathsf{x}\}$, as we also check the properties for the incoming data value $\star$ (again, recall that they consist in words over Σ^*), as our goal is to be able to reconstruct the input sequence of finite strings.

Note also that we impose no condition on $\star$ (the input value in $\mathbb{N}$ at this moment): it is simply ignored, as we only use this action to conduct the tests.

This translation of R-actions over $(\Sigma^*, <, \epsilon)$ to sequences of R'-actions over $(\mathbb{N}, <, 0)$ is extended to action sequences homomorphically. Since the resulting relation K is a morphism, it is in particular a rational relation. It remains to show that preserves feasibility.

First, assume that $\bar{a}$ is feasible by some data word $x = w_0 w_1 \cdots \in (\Sigma^*)^\omega$, and let $\nu_0 \nu_1 \ldots$ be a sequence of valuations that witnesses this fact, where, for all $i \geq 0$, $\nu_i : R \to \Sigma^*$, and $\nu_0 : r \mapsto \epsilon$. We translate it into the suitable sequence of *clen* values. Formally, for all $i \geq 0$, and for all $0 \leq j \leq k$, we let $n_i^j = \mathit{clen}(\nu_i(r_j), w_i)$. Additionally, $n_i^\star = \mathit{clen}(w_i, w_i) = |w_i|$, and $n_i^{\mathit{clen}} \in \mathbb{N}$ is some value whose choice does not matter (recall that the action corresponding to $\mathsf{tst}_{\mathit{clen}}$ is dedicated to checking that the input values indeed satisfy the prefix constraints and the string-compatibility conditions). Let us show by induction on i that there exists an action word $\bar{a}'$ in $K(\bar{a})$ such that $n_0^0 \ldots n_0^k n_0^\star n_0^{\mathit{clen}} n_1^0 \ldots n_1^k n_1^\star n_1^{\mathit{clen}} \ldots$ is compatible with $\bar{a}'$, and that it yields the following sequence of valuations: for all $i \geq 0$ and all $j \in \{0, \ldots, k, \star, \mathit{clen}\}$, $\nu'^j_i : \{r, s\} \in \binom{R}{2} \mapsto \mathit{clen}(\nu_i(r), \nu_i(s))$ and, for all $0 \leq m \leq k$, $\nu'^j_i(\{r_m, \mathsf{x}\}) = \begin{cases} \mathit{clen}(\nu_i(r_m), w_i) \text{ if } m \leq j \\ \nu'^k_{i-1}(r_m, \mathsf{x}) \text{ otherwise} \end{cases}$. Finally, for all $j \in \{0, \ldots, k, \star\}$, $\nu'^j_i(\{\mathsf{x}\}) = |w_{i-1}|$, and $\nu'^{\mathit{clen}}_i(\{\mathsf{x}\}) = |w_i|$.

For simplicity, we initialise the induction at -1 by picking $w_{-1} = \epsilon$, which makes the properties true. Now, assume that we have built $\bar{a}'$ up to step i_0. Thus, the current valuation is $\nu'^{\mathit{clen}}_{i_0}$, which is such that $\nu'^{\mathit{clen}}_{i_0} : \{r, s\} \mapsto \mathit{clen}(\nu_{i_0}(r), \nu_{i_0}(s))$. Additionally, $\nu'^{\mathit{clen}}_{i_0}(r, \mathsf{x}) \mapsto \mathit{clen}(r, \mathsf{x})$ for all $r \in R \cup \{\mathsf{x}\}$. Now, the R'-action word reads $n^0_{i_0+1} \ldots n^k_{i_0+1} n^\star_{i_0+1} n^{\mathit{clen}}_{i_0+1}$, as defined above. Since we set no conditions on $\mathsf{tst}_1, \ldots, \mathsf{tst}_k, \mathsf{tst}_\star$, we can pick suitable tests so that $n^0_{i_0+1} \ldots n^k_{i_0+1} n^\star_{i_0+1}$ satisfies them. Finally, $\mathsf{tst}_{\mathit{clen}}$ does not depend on $\star$, so any value of $n^{\mathit{clen}}_{i_0+1}$ is suitable. It remains to show that at this point, $\nu'^{\mathit{clen}}_{i_0+1}$ satisfies $\mathsf{tst}_{\mathit{clen}}$. By definition, it is a string-compatible counter valuation, since it is obtained from a string valuation. Thus, by [10, Proposition 2], it satisfies properties ψ_I, ψ_{II} and ψ_{III}. Moreover, ν_{i+1}, along with w_{i+1}, satisfies the *clen* constraints, so $\nu'^{\mathit{clen}}_{i_0+1}$ satisfies them as well. Thus, the property holds at step $i_0 + 1$.

Conversely, assume that some $\bar{a}' \in K(\bar{a})$ is feasible by some data word $n_0^0 \ldots n_0^k n_0^\star n_0^{\mathit{clen}} \cdot n_1^0 \ldots n_1^k n_1^\star n_1^{\mathit{clen}} \cdots \in \mathbb{N}^\omega$, along with a sequence of integer valuations $\nu'^0_0 \ldots \nu'^k_0 \nu'^\star_0 \nu'^{\mathit{clen}}_0 \cdot \nu'^0_1 \ldots \nu'^k_1 \nu'^\star_1 \nu'^{\mathit{clen}}_1 \ldots$. We build by induction on i a sequence of string valuations $\nu_i : R \to \Sigma^*$, along with a data word $x = w_0 w_1 \ldots$ which are compatible with $\bar{a}$, and such that for all $i \geq 0$, $\nu'^0_i : \binom{R \cup \{\mathsf{x}\}}{2} \to \mathbb{N}$ is a counter valuation that is string-compatible with $\nu_i[\star \leftarrow w_{i-1}]$, with the convention that $w_{-1} = \epsilon$.

Initially, $\nu_0 : R \mapsto \epsilon$ and $w_{-1} = \epsilon$, so $\nu'^0_0 : \{r, s\} \mapsto 0$ for all $r, s \in R \cup \{\mathsf{x}\}$, is string-compatible with ν_0.

Now, assume that we have built the ν_j up to step $i \geq 0$. By construction of $K(\bar{a}')$, we know that ν'^{l+1}_i satisfies $\psi_I \wedge \psi_{II} \wedge \psi_{III}$, as we ask that $\mathsf{tst}_{\mathit{clen}}$ implies them, and is such that for all $r, s \in R$, $\nu'^{\mathit{clen}}_i(\{r, s\}) = \mathit{clen}(\nu_i(r), \nu_i(s))$. Indeed, $\nu'^{\mathit{clen}}_{i\,|R} = \nu'^0_{i\,|R}$, as the values of the $\pi_{r,s}$ for $r, s \in R$ are left untouched before updating the registers when transitioning to ν'^0_{i+1}). In other words, the counters in ν'^{clen}_i indeed contain the length of the longest common prefixes of the strings of ν_i (in the terminology of [10], this means that $\nu'^{\mathit{clen}}_i \approx_R \nu_i$). Then, since ν'^{clen}_i additionally satisfies conditions ψ_I, ψ_{II} and ψ_{III} with regards to x, by [10, Lemma 6], we can construct a value for x that is consistent with ν_i. More precsiely, by applying [10, Lemma 6] to $X = \{\mathsf{x}\}$, we know that there exists a string w_i such that for all ${\nu'}_i^{clen} \approx_{R\cup\{\mathsf{x}\}} \nu_i[\star \leftarrow w_i]$, i.e. for all $r, s \in R \cup \{\mathsf{x}\}$ (note the addition of x), we have that ${\nu'}_i^{clen}(\{r, s\}) = clen(\nu_i(r), \nu_i(s))$, where $\nu_i(\star) = w_i$. Moreover, we know that the *clen* constraints are satisfied, since we encoded them in tst_{clen}. Thus, $w_0 \dots w_i w_{i+1}$ is compatible with $\bar{a}$ up to index $i + 1$, and the property holds at step $i + 1$. ◀