Logical Methods in Computer Science
Volume 17, Issue 1, 2021, pp. 22:1–22:25
https://lmcs.episciences.org/

Submitted Dec. 17, 2019
Published Mar. 18, 2021

SYNTHESIS OF DATA WORD TRANSDUCERS

LÉO EXIBARD [a,b], EMMANUEL FILIOT [b], AND PIERRE-ALAIN REYNIER [a]

[b] Université libre de Bruxelles, Brussels, Belgium
e-mail address: {leo.exibard,efiliot}@ulb.ac.be

[b] Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
e-mail address: pierre-alain.reynier@lis-lab.fr

Abstract. In reactive synthesis, the goal is to automatically generate an implementation from a specification of the reactive and non-terminating input/output behaviours of a system. Specifications are usually modelled as logical formulae or automata over infinite sequences of signals (ω-words), while implementations are represented as transducers. In the classical setting, the set of signals is assumed to be finite. In this paper, we consider data ω-words instead, i.e., words over an infinite alphabet. In this context, we study specifications and implementations respectively given as automata and transducers extended with a finite set of registers. We consider different instances, depending on whether the specification is nondeterministic, universal or deterministic, and depending on whether the number of registers of the implementation is given or not.

In the unbounded setting, we show undecidability for both universal and nondeterministic specifications, while decidability is recovered in the deterministic case. In the bounded setting, undecidability still holds for nondeterministic specifications, but can be recovered by disallowing tests over input data. The generic technique we use to show the latter result allows us to reprove some known result, namely decidability of bounded synthesis for universal specifications.

Introduction

Reactive synthesis is an active research domain whose goal is to design algorithmic methods able to automatically construct a reactive system from a specification of its admissible behaviours. Such systems are notoriously difficult to design correctly, and the main appealing idea of synthesis is to automatically generate systems that are *correct by construction*. Reactive systems are non-terminating systems that continuously interact with the environment in which they are executed, through input and output *signals*. At each time step,

Key words and phrases: Register Automata, Synthesis, Data words, Transducers.

This paper is the journal version of [EFR19]. ACM Classification: Theory of computation → Logic and verification; Theory of computation → Automata over infinite objects; Theory of computation → Transducers.

L. Exibard is funded by a FRIA fellowship from the F.R.S.-FNRS. E. Filiot is a research associate of F.R.S.-FNRS. He is supported by the ARC Project Transform Fédération Wallonie-Bruxelles and the FNRS CDR J013116F and MIS F451019F projects. P.-A. Reynier is partly funded by the DeLTA project (ANR–16–CE40–0007).

DOI:10.23638/LMCS-17(1:22)2021
© L. Exibard, E. Filiot, and P.-A. Reynier
Creative Commons

the system receives an input signal from a set In and produces an output signal from a set Out. An execution is then modelled as an infinite sequence alternating between input and output signals, i.e., an ω-word in $(\mathsf{In} \cdot \mathsf{Out})^\omega$. Classically, the sets In and Out are *assumed to be finite* and reactive systems are modelled as (sequential) *transducers*. Transducers are simple finite-state machines with transitions of type $\mathsf{States} \times \mathsf{In} \to \mathsf{States} \times \mathsf{Out}$, which, at any state, can process any input signal and deterministically produce some output signal, while possibly moving, again deterministically, to a new state. A *specification* is then a language $S \subseteq (\mathsf{In} \cdot \mathsf{Out})^\omega$ telling which are the acceptable behaviours of the system. It is also classically represented as an automaton, or as a logical formula then converted into an automaton. Some regular specifications may not be realisable by any transducer, and the *realisability problem* asks, given a regular specification S, whether there exists a transducer T whose behaviours satisfy S (i.e., are included in S). The synthesis problem asks to construct T if it exists.

A typical example of reactive system is that of a server granting requests from a *finite* set of clients C. Requests are represented as the set of input signals $\mathsf{In} = \{(r, i) \mid i \in C\} \cup \{\mathsf{idle}\}$ (client i requests the resource) and grants by the set of output signals $\mathsf{Out} = \{(g, i) \mid i \in C\} \cup \{\mathsf{idle}\}$ (server grants client i's request). A typical constraint to be imposed on such a system is that every request is eventually granted, which can be represented by the LTL formula $\bigwedge_{i \in C} G((r, i) \to F(g, i))$. The latter specification is realisable for instance by the transducer which outputs (g, i) whenever it reads (r, i) and idle whenever it reads idle.

It is well-known that the realisability problem is decidable for ω-regular specifications. It is ExpTime-complete when represented by parity automata [BL69, PR89, FJLW16]; and 2ExpTime-complete for LTL specifications [PR89]. Such positive results have triggered a recent and very active research interest in efficient symbolic methods and tools for reactive synthesis (see e.g. [BCJ18]). Extensions of this classical setting have been proposed to capture more realistic scenarios [BCJ18]. However, only a few works have considered infinite sets of input and output signals. In the previous example, the number of clients is assumed to be finite, and small. To the best of our knowledge, existing synthesis tools do not handle large alphabets, although it is more realistic to consider an unbounded (infinite) set of client identifiers, e.g. $C = \mathbb{N}$. The goal of this paper is to investigate how reactive synthesis can be extended to handle infinite sets of signals.

Data words are infinite sequences $x_1 x_2 \dots$ of labelled data, i.e., pairs (σ, d) with σ a label from a finite alphabet and d is a *data* from a countably infinite alphabet $\mathcal{D}$. They can naturally model executions of reactive systems over an infinite set of signals. Among other models, *register automata* are one of the main extensions of automata recognising languages of data words [KF94, Seg06]. They can use a finite set of registers in which to store data that are read, and to compare the current data with the content of some of the registers (in this paper, we allow comparison of equality). Likewise, transducers can be extended to *register transducers* as a model of reactive systems over data words: a register transducer is equipped with a set of registers, and when reading an input labelled data (σ, d), it can test d for equality with the content of some of its registers, and depending on the result of this test, deterministically assign some of its registers to d and output a finite label β along with the content of one of its registers. Its executions are then data words alternating between input and output labelled data, and register automata can thus be used to represent specifications, as languages of such data words.

Cannot output guessed data values

Contributions. We consider two classical semantics for register automata, nondeterministic and universal, both with a parity acceptance condition, which give two classes of register automata respectively denoted NRA and URA. We study the parity acceptance condition because it can express the other classical acceptance conditions; e.g., Büchi and co-Büchi can be expressed with a 2-colours parity condition. Since NRA are not closed under complement (already over finite data words), NRA and URA define incomparable classes of specifications. The request-grant specification, as defined above, can be generalised to an infinite number of clients, and it is then expressible by an URA [KMB18]: whenever a request is made by client i (labelled data (r, i)), universally trigger a run which stores i in some register and verifies that the labelled data (g, i) eventually occurs in the data word. In contrast, no NRA can define it. On the other hand, consider the specification S_0: "all input data but one are copied on the output, the missing one being replaced by some data which occurred before it", modelled as the set of data sequences $d_1 d_1 d_2 d_2 \dots d_i d_j d_{i+1} d_{i+1} \dots$ for all $i \geq 0$ and $j < i$ (finite labels are irrelevant and not represented). S_0 is not definable by any URA, as it would require to guess j, which can be arbitrarily smaller than i, but it is expressible by some NRA making this guess.

However, we show (unsurprisingly) that the realisability problem by register transducers of specifications defined by NRA is undecidable. The same negative result also holds for URA, solving an open question raised in [KMB18]. On the positive side, we show that decidability is recovered for deterministic (parity) register automata (DRA) in which the output is driven by the input (meaning that it is contained in some register). We call this class the DRA with input-driven outputs, denoted by $\mathsf{DRA_{ido}}$. One of the difficulties of register transducer synthesis is that the number of registers needed to realise the specification is, a priori, unbounded with regards to the number of registers of the specification. We show it is in fact not the case for $\mathsf{DRA_{ido}}$: any specification expressed as a $\mathsf{DRA_{ido}}$ with r registers is realisable by a register transducer iff it is realisable by a transducer with r registers.

A way to obtain decidability is to fix a bound k and to target register transducers with at most k registers. This setting is called *bounded synthesis* in [KMB18], which establishes that bounded synthesis is decidable in 2ExpTime for URA. We show that unfortunately, bounded synthesis is still undecidable for NRA specifications (even when targetting implementations with a single register). To recover decidability for NRA, we disallow equality tests on the input data and add a syntactic requirement which entails that on any accepted word, each output data is the content of some register which has been assigned an input data occurring before. This defines a subclass of NRA that we call (input) *test-free* NRA ($\mathsf{NRA_{tf}}$). $\mathsf{NRA_{tf}}$ can express how output data can be obtained from input data (by copying, moving or duplicating them), although they do not have the whole power of register automata on the input nor the output side. Note that the specification S_0 given before is $\mathsf{NRA_{tf}}$-definable. To show that bounded synthesis is decidable for $\mathsf{NRA_{tf}}$, we establish a generic transfer property characterising realisable data word specifications in terms of realisability of corresponding specifications over a finite alphabet, thus reducing to the well-known synthesis problem over a finite alphabet. Such property also allows us to reprove the result of [KMB18], with a rather short proof based on standard results from the theory of register automata, indicating that it might allow to establish decidability for other classes of data specifications. Our results are summarised in Table 1.

Related Work. As already mentioned, bounded synthesis of register transducers is considered in [KMB18] where it is shown to be decidable for URA. We reprove this result in a shorter way.

	$\mathsf{DRA_{ido}}$	NRA	URA	$\mathsf{NRA_{tf}}$
Bounded Synthesis	2ExpTime (Thm. 4.6)	Undecidable ($k \geq 1$) (Thm. 3.2)	2ExpTime ([KMB18] and Thm. 4.6)	2ExpTime (Thm. 4.11)
General Case	ExpTime-c (Thm. 3.7)	Undecidable (Thm. 3.1)	Undecidable (Thm. 3.3)	Open

Table 1. Decidability status of the problems studied. As observed in Corollary 3.8, the bounded synthesis for $\mathsf{DRA_{ido}}$ is in ExpTime if the target number of registers is larger than or equal to the number of registers of the specification.

Our proof bears some similarities with that of [KMB18], but it seems that our formulation benefits more from the use of existing results. The technique is also more generic and we instantiate it to $\mathsf{NRA_{tf}}$. $\mathsf{NRA_{tf}}$ correspond to the one-way, nondeterministic version of the expressive transducer model of [DH16], which however does not consider the synthesis problem.

The synthesis problem over infinite alphabets is also considered in [ESK14], in which data represent identifiers and specifications (given as particular automata close to register automata) can depend on equality between identifiers. However, the class of implementations is very expressive: it allows for unbounded memory through a queue data structure. The synthesis problem is shown to be undecidable and a sound but incomplete algorithm is given.

Finally, classical reactive synthesis has strong connections with game theory on finite graphs. Some extension of games to infinite graphs whose vertices are valuations of variables in an infinite data domain have been considered in [FP18]. Such games are shown to be undecidable and a decidable restriction is proposed, which however does not seem to match our context.

1. Data Words and Register Automata

For a (possibly infinite) set S, we denote by S^ω the set of infinite words over this alphabet. For $1 \leq i \leq j$, we let $u[i{:}j] = u_i u_{i+1} \dots u_j$ and $u[i] = u[i{:}i]$ the ith letter of u. For $u, v \in S^\omega$, we define their *interleaving* $\langle u, v \rangle = u[1]v[1]u[2]v[2]\dots$

1.1. **Data Words.** Let Σ be a finite alphabet and $\mathcal{D}$ a countably infinite set, denoting, all over this paper, a set of elements called *data*. We also distinguish an (arbitrary) data value $\mathsf{d}_0 \in \mathcal{D}$. Given a set R, let τ_0^R be the constant function defined by $\tau_0^R(r) = \mathsf{d}_0$ for all $r \in R$. A *labelled data* (or l-data for short) is a pair $x = (\sigma, d) \in \Sigma \times \mathcal{D}$, where σ is the *label* and d the *data*. We define the projections $\mathsf{lab}(x) = \sigma$ and $\mathsf{dt}(x) = d$. A *data word* over Σ and $\mathcal{D}$ is an infinite sequence of labelled data, i.e. a word $w \in (\Sigma \times \mathcal{D})^\omega$. We extend the projections lab and dt to data words naturally, i.e. $\mathsf{lab}(w) \in \Sigma^\omega$ and $\mathsf{dt}(w) \in \mathcal{D}^\omega$. We denote the set of data words over Σ and $\mathcal{D}$ by $\mathsf{DW}(\Sigma, \mathcal{D})$ (DW when clear from the context). A *data word language* is a subset $L \subseteq \mathsf{DW}(\Sigma, \mathcal{D})$. Note that in this paper, data words are infinite, otherwise they are called *finite data words*, and we denote by $\mathsf{DW}_f(\Sigma, \mathcal{D})$ the set of finite data words.

1.2. **Register Automata.** Register automata are automata recognising data word languages. They were first introduced in [KF94] as finite-memory automata. Here, we define them in a spirit close to [LTV15], but over infinite words, with a parity acceptance condition. The current data can be compared for equality with the register contents via tests. Our tests are symbolic and defined via Boolean formulas of the following form. Given R a set of registers, a *test* is a formula ϕ satisfying the following syntax:

$$\phi \ ::= \ \top \mid \bot \mid r^{=} \mid r^{\neq} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi$$

where $r \in R$. Given a valuation $\tau : R \to \mathcal{D}$, a test ϕ and a data d, we denote by $\tau, d \models \phi$ the satisfiability of ϕ by d in valuation τ, defined as $\tau, d \models r^{=}$ if $\tau(r) = d$ and $\tau, d \models r^{\neq}$ if $\tau(r) \neq d$. The Boolean combinators behave as usual. We denote by Tst_R the set of (symbolic) tests over R.

Definition 1.1. A *register automaton* (RA) is a tuple $\mathcal{A} = (\Sigma, \mathcal{D}, Q, q_0, \delta, R, c)$, where:

- Σ is a finite alphabet of *labels*, $\mathcal{D}$ is an infinite alphabet of *data*
- Q is a finite set of states and $q_0 \in Q$ is the initial state
- R is a finite set of *registers*. We denote $\mathsf{Asgn}_R = 2^R$.
- $c : Q \to \{1, \dots, d\}$, where $d \in \mathbb{N}$ is the number of *priorities*, is the *colouring function*, used to define the acceptance condition
- $\delta \subseteq Q \times \Sigma \times \mathsf{Tst}_R \times \mathsf{Asgn}_R \times Q$ is a set of transitions.

A transition $(q, \sigma, \phi, \mathsf{asgn}, q')$ is also written $q \xrightarrow[\mathcal{A}]{\sigma,\phi,\mathsf{asgn}} q'$. We may omit $\mathcal{A}$ in the latter notation. Intuitively such transition means that on input (σ, d) in state q the automaton:

(1) checks that ϕ is satisfied by the current register contents and the current data
(2) assigns d to all the registers in asgn (asgn might be empty)
(3) transitions to state q'.

$\mathcal{A}$ is said to be *deterministic* if the tests are mutually exclusive, i.e., for any two distinct transitions of the form $q \xrightarrow{\sigma,\phi,\mathsf{asgn}} q'$ and $q \xrightarrow{\sigma',\phi',\mathsf{asgn}'} q''$, then either $\sigma \neq \sigma'$ or $\phi \wedge \phi'$ is not satisfiable. The automaton $\mathcal{A}$ is said to be *complete* if for any given state q, any label σ, any data d and any register valuation τ, there exists a transition $q \xrightarrow{\sigma,\phi,\mathsf{asgn}} q' \in \delta$ such that $\tau, d \models \phi$.

1.3. **Configurations and Runs.** A configuration is a pair $(q, \tau) \in Q \times (R \to \mathcal{D})$. Fix a transition $t = p \xrightarrow{\sigma,\phi,\mathsf{asgn}} p'$. We say that (q, τ) enables t on reading (σ', d) if $q = p$, $\sigma' = \sigma$ and $\tau, d \models \phi$. Let $\mathrm{next}(\tau, \mathsf{asgn}, d)$ be the valuation τ' defined by $\tau'(i) = d$ if $i \in \mathsf{asgn}$, and $\tau'(i) = \tau(i)$ otherwise. We extend this notation to configurations as follows: if $\gamma = (q, \tau)$ enables t on input (σ, d), the *successor* configuration of (q, τ) by t on input (σ, d) is $\mathrm{next}(\gamma, \mathsf{asgn}, d, t) = (p', \mathrm{next}(\tau, \mathsf{asgn}, d))$. We also write $\mathrm{next}(\gamma, t, \sigma, d)$ to denote the successor of (q, τ) by transition t when (q, τ) enables t on input (σ, d). The *initial configuration* is (q_0, τ_0^R). Then, a *run* over a data word $(\sigma_1, d_1)(\sigma_2, d_2)\dots$ is an infinite sequence of transitions $t_0 t_1 \dots$ such that there exists a sequence of configurations $\gamma_0\gamma_1 \cdots = (q_0, \tau_0)(q_1, \tau_1)\dots$ such that γ_0 is initial and for all $i \geq 0$, $\gamma_{i+1} = \mathrm{next}(\gamma_i, t_i, \sigma_i, d_i)$. With a run ρ, we associate its sequence of states $\mathsf{states}(\rho) = q_0 q_1 \dots$

1.4. **Languages Defined by RA.** Given a run ρ, we denote, by a slight abuse of notation, $c(\rho) = \max\{j \mid c(q_l) = j \text{ for infinitely many } q_l \in \mathsf{states}(\rho)\}$ the maximum color that occurs infinitely often in ρ. Then, in the parity acceptance condition, ρ is accepting whenever $c(\rho)$ is even. We consider two dual semantics for RA: nondeterministic (N) and universal (U). Given a RA A, depending on whether it is considered nondeterministic or universal, it recognises $L_N(A) = \{w \mid \text{there exists an accepting run } \rho \text{ on } w\}$ or $L_U(A) = \{w \mid \text{all runs } \rho \text{ on } w \text{ are accepting}\}$. Note that those semantics are dual: for a RA A, by letting $\overline{A}$ be a copy of A with colouring function $\overline{c} : q \mapsto c(q) + 1$, we have that $L_U(\overline{A}) = \overline{L_N(A)}$.

We denote by NRA (resp. URA) the class of register automata interpreted with a nondeterministic (resp. universal) parity acceptance condition, and given $A \in \mathsf{NRA}$ (resp. $A \in \mathsf{URA}$), we write $L(A)$ instead of $L_N(A)$ (resp. $L_U(A)$). We also denote by DRA the class of deterministic parity register automata.

2. Synthesis of Register Transducers

2.1. **Specifications, Implementations and the Realisability Problem.** Let $\Sigma_{\mathbb{i}}$ and $\Sigma_{\mathbb{o}}$ be two finite alphabets of labels, and $\mathcal{D}$ a countable set of data. A *relational data word* is an element of $w \in [(\Sigma_{\mathbb{i}} \times \mathcal{D}) \cdot (\Sigma_{\mathbb{o}} \times \mathcal{D})]^\omega$. Such a word is called relational as it defines a pair of data words in $\mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D}) \times \mathsf{DW}(\Sigma_{\mathbb{o}}, \mathcal{D})$ through the following projections. If $w = x^1_{\mathbb{i}} x^1_{\mathbb{o}} x^2_{\mathbb{i}} x^2_{\mathbb{o}} \dots$, we let $\mathsf{inp}(w) = x^1_{\mathbb{i}} x^2_{\mathbb{i}} \dots$ and $\mathsf{out}(w) = x^1_{\mathbb{o}} x^2_{\mathbb{o}} \dots$ We denote by $\mathsf{RW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D})$ (just RW when clear from the context) the set of relational data words. A *specification* is simply a language $S \subseteq \mathsf{RW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D})$. An *implementation* is a total function $I : (\Sigma_{\mathbb{i}} \times \mathcal{D})^* \to \Sigma_{\mathbb{o}} \times \mathcal{D}$. From I, we define another function $f_I : \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D}) \to \mathsf{DW}(\Sigma_{\mathbb{o}}, \mathcal{D})$ which, with an input data word $w_{\mathbb{i}} = x^1_{\mathbb{i}} x^2_{\mathbb{i}} \dots \in \Sigma_{\mathbb{i}} \times \mathcal{D}$, associates the output data word $f_I(w_{\mathbb{i}}) = x^1_{\mathbb{o}} x^2_{\mathbb{o}} \dots$ such that $\forall i \geq 1$, $x^i_{\mathbb{o}} = I(x^1_{\mathbb{i}} \dots x^{i-1}_{\mathbb{i}})$. I also defines a language of relational data words $L(I) = \{\langle w_{\mathbb{i}}, f_I(w_{\mathbb{i}})\rangle \mid w_{\mathbb{i}} \in \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})\}$.

We say that I *realises* S when $L(I) \subseteq S$, and that S is *realisable* if there exists an implementation realising it. Note that since f_I is a total function, we have that if S is realisable, then in particular its domain is total, i.e. for all $w_{\mathbb{i}} \in \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$, there exists $w_{\mathbb{o}} \in \mathsf{DW}(\Sigma_{\mathbb{o}}, \mathcal{D})$ such that $\langle w_{\mathbb{i}}, w_{\mathbb{o}}\rangle \in S$. Therefore, any specification whose domain is not total is not realisable according to this definition. For a discussion on this definition, see Section 5.

The *realisability problem* consists, given a (finite representation of a) specification S, in checking whether S is realisable. In general, we parameterise this problem by classes of specifications $\mathcal{S}$ and of implementations $\mathcal{I}$, defining the $(\mathcal{S}, \mathcal{I})$-realisability problem, denoted $\textsc{Real}(\mathcal{S}, \mathcal{I})$. Given a specification $S \in \mathcal{S}$, it asks whether S is realisable by some implementation $I \in \mathcal{I}$. We now introduce the classes $\mathcal{S}$ and $\mathcal{I}$ we consider.

2.2. **Specification Register Automata.** In this paper, we consider specifications defined by register automata (hence alternately reading input and output labelled data). We assume that the set of states is partitioned into $Q_{\mathbb{i}}$ (called input states, reading only labels in $\Sigma_{\mathbb{i}}$) and $Q_{\mathbb{o}}$ (called output states, reading only labels in $\Sigma_{\mathbb{o}}$), where $q_0 \in Q_{\mathbb{i}}$, and such that the transition relation δ alternates between these two sets, i.e.

$$\delta \subseteq \bigcup_{\alpha = \mathbb{i}, \mathbb{o}} (Q_\alpha \times \Sigma_\alpha \times \mathsf{Tst}_R \times \mathsf{Asgn}_R \times Q_{\overline{\alpha}}),$$

where $\overline{\mathbb{i}} = \mathbb{o}$ (resp. $\overline{\mathbb{o}} = \mathbb{i}$). We denote by DRA (resp. NRA, URA) the class of specifications defined by deterministic (resp. nondeterministic, universal) parity register automata.

Example 2.1. Remember the setting described in the introduction of a server granting requests from an unbounded set of clients C. The input (resp. output) finite alphabets are $\Sigma_{\mathbb{i}} = \{\mathsf{req}, \mathsf{idle}\}$ and $\Sigma_{\mathbb{o}} = \{\mathsf{grt}, \mathsf{idle}\}$, while the set of data is any countably infinite set $\mathcal{D}$ containing C. Without loss of generality, $C \subseteq \mathbb{N}$ is a set of client ids, so we can take $\mathcal{D} = \mathbb{N}$. Then, as stated in the introduction, the specification that for all $i \in C$, every request of client i is eventually granted can be expressed with the URA of Figure 1.

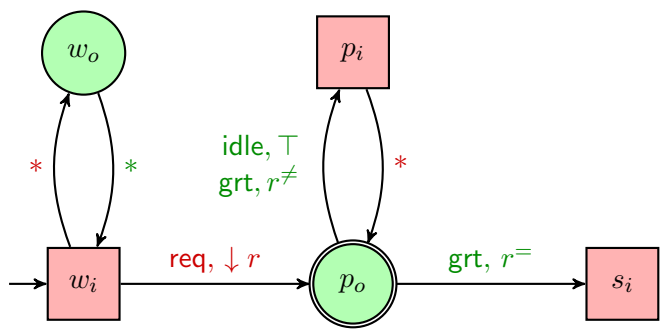


FIGURE 1. A universal register automaton checking that every request is eventually granted. Input is in red (states are squares), output is in green (states are circles). Finite labels are sans serif. All states have priority 0, except the doubly circled state p_o, which has priority 1. This corresponds to a co-Büchi acceptance condition with rejecting state p_o. The automaton always loops between w_i and w_o (the $*$ symbol means that the transition is taken, no matter the labelled-data received). Whenever it receives a request as input, it universally spawns a run which stores the corresponding id in its single register r (depicted as $\downarrow r$), and transitions to p_0. Then, it loops between p_i and p_o while it does not receive the corresponding grant, with matching id, as output (i.e. either reads idle or receives a grant with wrong id: $d \neq r$). When it receives a grant with the right id ($d = r$), it transitions to s_i, then the run dies at the next step (which favors acceptance in the universal semantics).

2.3. **Register Transducers As Implementations.** We consider implementations represented as transducers processing data words. A *register transducer* is a tuple $T = (\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, Q, q_0, \delta, R)$ where Q is a finite set of states with initial state q_0, R is a finite set of registers, and $\delta : Q \times \Sigma_{\mathbb{i}} \times \mathsf{Tst}_R \to \mathsf{Asgn}_R \times \Sigma_{\mathbb{o}} \times R \times Q$ is the transition function (as before, $\mathsf{Asgn}_R = 2^R$), assumed to be complete in the sense that, as for RA, for every state q and label $\sigma_{\mathbb{i}}$, for every data d and register valuation τ, there exists a transition $\delta(q, \sigma_{\mathbb{i}}, \phi) = (\mathsf{asgn}, \sigma_{\mathbb{o}}, r, q')$ such that $\tau, d \models \phi$. When processing an l-data $(\sigma_{\mathbb{i}}, d)$, T compares d with the content of some of its registers, and depending on the result, moves to another state, stores d in some registers, and outputs some label in $\Sigma_{\mathbb{o}}$ along with the content of some register $r \in R$.

Let us formally define the semantics of a register transducer T, as an implementation I_T. First, for a finite input data word $w = (\sigma_{\mathbb{I}}^1, d_{\mathbb{I}}^1) \dots (\sigma_{\mathbb{I}}^n, d_{\mathbb{I}}^n)$ in $(\Sigma_{\mathbb{I}} \times \mathcal{D})^*$, we denote by (q_i, τ_i) the ith configuration reached by T on w, where (q_0, τ_0) is initial and for all $0 < i < n$, (q_i, τ_i) is the unique configuration such that there exists a transition $\delta(q_{i-1}, \sigma_{\mathbb{I}}^i, \phi) = (\mathsf{asgn}, \sigma_{\mathbb{O}}, r, q_i)$ such that $\tau_{i-1}, d_{\mathbb{I}}^i \models \phi$ and $\tau_i = \mathrm{next}(\tau_{i-1}, d_{\mathbb{I}}^i, \mathsf{asgn})$. We let $(\sigma_{\mathbb{O}}^i, d_{\mathbb{O}}^i) = (\sigma_{\mathbb{O}}, \tau_i(r))$ and $I_T(w) = (\sigma_{\mathbb{O}}^n, d_{\mathbb{O}}^n)$. Then, we denote $f_T = f_{I_T}$ and $L(T) = L(I_T)$. Note that if T is interpreted as a DRA with exactly one transition per output state and whose states are all accepting (i.e. have even maximal parity 0), then $L(I_T)$ is indeed the language of such register automaton. We denote by $\mathsf{RT}[k]$ the class of implementations defined by register transducers with at most k registers, and by $\mathsf{RT} = \bigcup_{k \geq 0} \mathsf{RT}[k]$ the class of implementations defined by register transducers.

Example 2.2. Consider again the specification of Example 2.1. Such specification is realisable for instance by the transducer which outputs (grt, i) whenever it reads (req, i) and (idle, d) (d does not matter) whenever it reads idle, which is depicted in Figure 2.

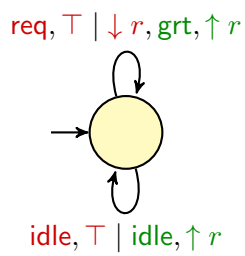


Figure 2. A register transducer immediately granting each request. The notations are the same as in Figure 1. Additionally, here, $\uparrow r$ means that the transducer outputs the content of r.

2.4. **Synthesis from Data-Free Specifications.** If in the latter definitions of the synthesis problem, one considers specifications defined by RA with no registers (i.e. parity automata), and implementations defined by RT with no registers, then the data in data-words can be ignored and we are back to the classical reactive synthesis setting, for which important results are known:

Theorem 2.3 [BL69]**.** *The realisability problem of (data-free) specifications given as (register-free) nondeterministic parity automata by (register-free) transducers is* ExpTime*-complete.*

Proof. The upper bound was first established in [BL69] and [PR89]. Hardness is folklore, but a proof in the particular case of finite words (easily adapted to the ω-word setting) can be found in [FJLW16, Proposition 6]. □

3. Unbounded Synthesis

In this section, we consider the unbounded synthesis problem $\mathrm{Real}(\mathsf{RA}, \mathsf{RT})$. Thus, we do not fix a priori the number of registers of the implementation.

3.1. **Undecidability Results.** Let us first consider the case of NRA and URA, which are, in our setting, the most natural devices to express data word specifications. Unfortunately, the two corresponding problems happen to be undecidable:

Theorem 3.1. *$\textsc{Real}(\mathsf{NRA}, \mathsf{RT})$ is undecidable.*

Proof. We reduce the problem from the universality of NRA over finite words, which is undecidable [NSV04]. Let A be a (finite data-word) NRA. Let S be a specification which first reads some finite data word w, then a separator # (its associated data is arbitrary and not represented), then allows for swapping the first and second l-data on any input read later on, while also allowing to behave like the identity whenever $w \in L(A)$. S is also equal to the identity over any word not containing # so that its domain is total. Formally, let $S = S_1 \cup S_2 \cup T$, where:

$$S_1 = \left\{ (w\#(\sigma_1, d_1)(\sigma_2, d_2)u, w\#(\sigma_2, d_1)(\sigma_1, d_2)u) \;\middle|\; \begin{array}{c} d_1, d_2 \in \mathcal{D}, \sigma_1, \sigma_2 \in \Sigma \\ w \in \mathsf{DW}_f, u \in \mathsf{DW} \end{array} \right\}$$
$$S_2 = \{(w\#u, w\#u) \mid w \in L(A), u \in \mathsf{DW}\}$$
$$T = \{(w, w) \mid w \notin \mathsf{DW}_f\#\mathsf{DW}\}$$

S is definable by a NRA running over relational data words, because each component is and NRA are closed under union. Recognising the interversion of the first two labels σ_1 and σ_2 after the # in S_1 is easily done using nondeterminism, and the behaviour on data is the identity, so S_1 is NRA-definable. Then, emulating the identity over some NRA-definable domain is easy, so S_2 and T are also NRA-definable.

Now, if A is universal, ie $L(A) = \mathsf{DW}_f$, then the identity $\mathrm{id}_{\mathsf{DW}}$ over DW realises S, since then $\mathrm{id}_{\mathsf{DW}} \subseteq S$ and has total domain. Conversely, if $L(A) \subsetneq \mathsf{DW}_f$, assume by contradiction that S is realisable by a register transducer I. Let $w \in \mathsf{DW}_f \setminus L(A)$. Then, for any $(\sigma_1, d_1)(\sigma_2, d_2)u \in \mathsf{DW}$, we must have $I(w\#(\sigma_1, d_1)(\sigma_2, d_2)u) = w\#(\sigma_2, d_1)(\sigma_1, d_2)u$; but this implies guessing the second label while having only read the first one, which is not doable by any transducer as long as $\sigma_1 \neq \sigma_2$. □

what about Real(1-NRA, RT)?

Actually, we can observe that such undecidability proof extends to $\textsc{Real}(\mathsf{NRA}, \mathsf{RT}[1])$, and to all $\textsc{Real}(\mathsf{NRA}, \mathsf{RT}[k])$ for $k \geq 1$. Indeed, A is universal iff S is realisable by the identity over data words, which is implementable using a 1-register transducer:

Theorem 3.2. *For all $k \geq 1$, $\textsc{Real}(\mathsf{NRA}, \mathsf{RT}[k])$ is undecidable.*

Now, we can show that the unbounded synthesis problem is also undecidable for URA, answering a question left open in [KMB18].

and 1-URA?

Theorem 3.3. *$\textsc{Real}(\mathsf{URA}, \mathsf{RT})$ is undecidable.*

Proof. We present a reduction to our synthesis problem from the emptiness problem of URA over finite words. The latter is undecidable by a direct reduction from the universality problem of NRA, which is undecidable by [NSV04].

First, consider the relation $S_1 = \{(u\#v, u\#w) \mid u \in \mathsf{DW}_f, v \in \mathsf{DW}$, each data of u appears infinitely often in $w\}$. S_1 is recognised by a 1-register URA which, upon reading a data d in u, stores it in its register and checks that it appears infinitely often in w by visiting a state with maximal parity 2 every time it sees d (all other states have parity 1). Note that for all $k \geq 1$, $S_1 \cap \{(u\#v, u\#w) \mid u \in \mathsf{DW}_f, v, w \in \mathsf{DW}$ and u has at most k distinct data$\}$ is realisable by a k-register transducer: on reading u, store each distinct data in one register, and after the # output them in turn in a round-robin fashion. However, S_1 is not realisable:

on reading the # separator, any implementation must have all the data of u in its registers, but the number of such data is not bounded (u can have pairwise distinct data and be of arbitrary length).

Then, let A be a URA over finite data words. Consider the specification $S = S_1 \cup S_2 \cup T$, where $S_2 = \{(u\#v, u\#w\#(a, \mathsf{d}_0)^\omega) \mid u \in \mathsf{DW}_f, v \in \mathsf{DW}, w \in L(A)\}$ and $T = \{(u, w) \mid u \notin \mathsf{DW}_f\#\mathsf{DW}, w \in \mathsf{DW}\}$. S has total domain, and is recognisable by a URA. Indeed, URA are closed under union, by the same product construction as for the intersection of NRA [KF94], and each part is URA-recognisable: S_1 is, as described above, S_2 is by simulating A on the output to check $w \in L(A)$ then looping over (a, d_0), and T simply checks a regular property.

Now, if $L(A) \neq \varnothing$, let $w \in L(A)$ and let $D_w = \{d_1, \ldots, d_k\}$ be the set of data distinct from d_0 that occur in w. As a consequence of the closure under automorphisms of register automata [KF94, Proposition 2], we have: for any set $D \subseteq \mathcal{D}$ such that $|D| \geq k$, and for any injection $\pi : D_w \cup \{\mathsf{d}_0\} \to D \cup \{\mathsf{d}_0\}$ such that $\pi(\mathsf{d}_0) = \mathsf{d}_0$, by extending π to a morphism $\widehat{\pi}$ over data words in the usual way (and behaving as the identity over the finite labels), $\widehat{\pi}(w) \in L(A)$. Indeed, as register automata can only test for equality, acceptance is determined by the equality relations between the different data of the input, so we can rename them (with the exception of d_0, which is a distinguished data).

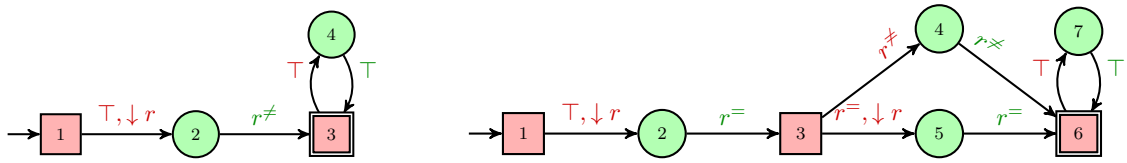
Then, S is realisable by a register transducer I with $k + 2$ registers. While it has not read a #, I reads its input u and outputs it along the way, using one register to store the current data and output it immediately. Meanwhile, it also stores the first k distinct data of u in its registers. Its last register is used to keep d_0 in memory. If there is no # in the input, then $I(u) = u$, so $(u, I(u)) \in T$. Now, if some # is read, I outputs # (along with an arbitrary data), and there are two cases: if the number of data in u is lower than or equal to k, I realises S_1, as described above. Otherwise, let $D_u = \{e_1, \ldots, e_l\}$ be the set of data of u distinct from d_0, indexed by order of appearance ($l \geq k$). Then, let $\pi : D_w \cup \{\mathsf{d}_0\} \to D_u \cup \{\mathsf{d}_0\}$ be such that for all $1 \leq i \leq k, \pi(d_i) = e_i$ and $\pi(\mathsf{d}_0) = \mathsf{d}_0$: π is injective. Now, I can output $\widehat{\pi}(w)\#(a, \mathsf{d}_0)^\omega$ since it stored $\{e_1, \ldots, e_k\}$ in its registers, hence realising S_2. Conversely, if $L(A) = \varnothing$, then S is not realisable. If it were, $S \cap \mathsf{DW}_f\#\mathsf{DW} = S_1$ would be too, as a regular domain restriction, but we have seen above that this is not the case. Thus, S is realisable iff $L(A) \neq \varnothing$. □

3.2. **A Decidable Subclass: $\mathsf{DRA}_{\mathsf{ido}}$.** However, we show that restricting to DRA allows to recover decidability, modulo one additional assumption, namely that the output data of a transition has to be the content of some register. We formally define this class as follows:

Definition 3.4 ($\mathsf{DRA}_{\mathsf{ido}}$)**.** Let $\mathcal{A} = (\Sigma, \mathcal{D}, Q, q_0, \delta, R, c)$ be a DRA. We say that $\mathcal{A}$ is with *input-driven outputs* if for any output transition $p \xrightarrow{\sigma,\phi,\mathsf{asgn}} q$, the test ϕ is of the form $r^=$ for some $r \in R$. We denote by $\mathsf{DRA}_{\mathsf{ido}}$ the class of DRA with input-driven outputs.

Such assumption rules out pathological, and to our opinion uninteresting and technical cases stemming from the asymmetry between the class of specifications and implementations. E.g., consider the single-register DRA in Fig. 3a (finite labels are arbitrary and not depicted). It starts by reading one input data d and stores it in r, asks that the corresponding output data is different from the content d of r, then accepts any output over any input (transitions $\top$ are always takeable). It is not realisable because transducers necessarily output the content of some register (hence producing a data which already appeared). On the other hand, having tests of the form $\phi = r^{\neq}$ for instance does not imply unrealisability, as shown

by the DRA of Fig. 3b: it starts by reading one data d_1, asks to copy it on the output, then reads another data d_2, and requires that the output is either distinct from d_1 or equal to it, depending on whether $d_2 \neq d_1$. It happens that such specification is realisable by the identity.



(A) An unrealisable DRA. (B) A similar DRA, suprisingly realisable.

FIGURE 3. Pathological DRA specifications.

We reduce the realisability of $\mathsf{DRA}_{\mathsf{ido}}$-specifications to solving a finite parity game. To ease its construction, we first need to confer additional properties to the specification automaton.

A RA A is said to be *locally concretisable* if for every finite sequence of transitions $\rho = t_1 \dots t_n$, for every finite data word $w \in \mathsf{DW}_f$ such that ρ is a partial run of A on w, we have that for all transitions $t \in \delta$ which are compatible with ρ (i.e. such that the source state of t is equal to the end state of ρ), there exists $d \in \mathcal{D}$ such that ρt is a partial run of A on wd. Note in particular that when ρ is not a partial run, such condition trivially holds.

We say that a RA A is in *good form* if

(1) it is locally concretisable
(2) it is complete on its input states
(3) its tests ϕ are maximally consistent conjunctions of atoms
(4) any transition t whose test is different from $\bigwedge_{r \in R} r^{\neq}$ does not conduct an assignment ($\mathsf{asgn} = \varnothing$)

Lemma 3.5. *For all* RA A*, there exists an equivalent* RA A' *in good form with exponentially many more states and transitions, and the same number of priorities and registers. Moreover if A is a* $\mathsf{DRA}_{\mathsf{ido}}$*, so is A'.*

Proof. Let $A = (\Sigma, \mathcal{D}, Q, q_0, \delta, R, c)$ be a RA. First, we can assume that A is complete on its input states: add two sink states $s_{\mathbb{i}}$ and $s_{\mathbb{o}}$ with transitions $(s_{\mathbb{i}}, \sigma_{\mathbb{i}}, \top, \varnothing, s_{\mathbb{o}})$ and $(s_{\mathbb{o}}, \sigma_{\mathbb{o}}, r^{=}, \varnothing, s_{\mathbb{i}})$ for all $\sigma_{\mathbb{i}} \in \Sigma_{\mathbb{i}}, \sigma_{\mathbb{o}} \in \Sigma_{\mathbb{o}}, r \in R$, each with odd priority $c(s_{\mathbb{i}}) = c(s_{\mathbb{o}}) = 1$. Then, for all $q_{\mathbb{i}} \in Q_{\mathbb{i}}$, and all finite label $\sigma_{\mathbb{i}} \in \Sigma_{\mathbb{i}}$, add a transition $q_{\mathbb{i}} \xrightarrow{\sigma_{\mathbb{i}}, \psi, \varnothing} s_{\mathbb{o}}$ where $\psi = \neg \bigvee_{q_{\mathbb{i}} \xrightarrow{\sigma_{\mathbb{i}}, \phi, \mathsf{asgn}} q_{\mathbb{o}}} \phi$ is a test which is satisfied by a data if and only if such data satisfies no other possible test. This does not affect determinism nor the recognised language (as each added state has odd priority), and preserves the fact of being ido.

Now, we enrich the states with information on the equalities between registers in the current register valuation. Formally, we define constraints[1] as equivalence relations on R. In the following, we denote by $\mathsf{ER}(R)$ the set of equivalence relations on R. Given a valuation τ of registers in R, we can associate to it an equivalence relation on R in the natural way (two registers $r, r' \in R$ are equivalent iff $\tau(r) = \tau(r')$). We denote it by $[\tau]$. We use the letter C to denote an element of $\mathsf{ER}(R)$, and we call it a constraint.

[1]The notion of *constraint* is pervasive in the study of registers automata, e.g. to recognise the projection over finite labels.

We let $A' = (\Sigma, \mathcal{D}, Q', q'_0, \delta', R, c')$ be defined as follows:

- $Q' = Q \times \mathsf{ER}(R)$
- $q'_0 = \big(q_0, [\tau_0^R]\big)$
- $c'(q, C) = c(q)$, for every $(q, C) \in Q \times \mathsf{ER}(R)$
- δ' will be defined in the sequel.

Given a constraint C, and a set $E \subseteq R$ corresponding to an equivalence class of C, we define a test corresponding to a maximally consistent conjunction of equalities and inequalities: $\alpha_E = \bigwedge_{r \in E} r^= \wedge \bigwedge_{r \notin E} r^{\neq}$. A data value satisfies this test iff it is equal to the (common) value stored in registers of R. We also consider the test $\alpha_\varnothing = \bigwedge_{r \in R} r^{\neq}$ which corresponds to the case of a fresh data value, i.e. a data value distinct from all the values stored in registers.

Consider a transition $(p, \sigma, \phi, \mathsf{asgn}, q) \in \delta$. Given a formula α_E as defined above, one can decide whether the formula $\alpha_E \Rightarrow \phi$ is valid or not. If this is the case, then we add the following transition to δ':

$$(p, C) \xrightarrow{\sigma, \alpha_E, \mathsf{asgn}} (q, C')$$

where C' is defined as follows: two registers r, r' are in relation with respect to C' if and only if one of the following cases holds:

- they are in relation in C, and not in asgn
- they are both in asgn
- r belongs to E and r' belongs to asgn, or vice versa.

First, observe that since A is complete on its input states, so is A' and property (2) holds. Moreover, by definition, A' satisfies property (3).

Now, one can show by induction on the length n of the partial run that every partial run $\rho = t_1 \dots t_n$ of A' over some finite data word $w \in \mathsf{DW}_f$ reaching some configuration $((p, C), \tau)$ satisfies $C = [\tau]$. Thus, for every run of A', by denoting $\{((q_i, C_i), \tau_i)\}_{i \in \mathbb{N}}$ its sequence of configurations, we have $C_i = [\tau_i]$.

Additionally, for each run of A, we can build a run of A' in a deterministic manner: let $\rho = t_1 t_2 \dots$ be a run of A over some data word $w = (\sigma_1, d_1)(\sigma_2, d_2) \dots$, where for all $i \in \mathbb{N}$, $t_{i+1} = q_i \xrightarrow{\sigma_i, \phi_i, \mathsf{asgn}_i} q_{i+1}$ and let $\{(q_i, \tau_i)\}_{i \in \mathbb{N}}$ be its sequence of configurations. Correspondingly, let $\rho' = t'_1 t'_2 \dots$, where for each $i \in \mathbb{N}$ $t'_{i+1} = (q_i, C_i) \xrightarrow[A']{\sigma_i, \alpha_{E_i}, \mathsf{asgn}_i} (q_{i+1}, C_{i+1})$, with $C_i = [\tau_i]$ and $E_i = \{r \in R \mid \tau_i(r) = d_i\}$. Then, again by induction, we can show that ρ' is a run of A' over w, whose sequence of configurations is $\{((q_i, C_i), \tau_i)\}_{i \in \mathbb{N}}$. Moreover, ρ' is accepting if and only if ρ is accepting, since $c'(q_i, C_i) = c(q_i)$. Reciprocally, every run ρ' of A' can be projected to a run of A by removing the C_i, and this preserves acceptance. Overall, $L(A) = L(A')$.

Now, let $\rho = t_1 \dots t_n$ be a partial run of A' over some finite data word $w \in \mathsf{DW}_f$ ending in some configuration $((q, C), \tau)$; recall that $C = [\tau]$. Let $t = q \xrightarrow{\sigma, \alpha_E, \mathsf{asgn}} q'$ be a transition compatible with ρ, i.e. such that q is the end state of ρ. If $E = \varnothing$, then $\alpha_E = \bigwedge_{r \in R} r^{\neq}$, so any $d \in \mathcal{D} \backslash \tau(R)$ (where $\tau(R)$ denotes the image of R by τ) is such that $\tau, d \models \alpha_E$. If $E \neq \varnothing$, then by construction E corresponds to an equivalence class of C, so $\forall r, r' \in E, \tau(r) = \tau(r')$ and $\forall r \in E, \forall r' \notin E, \tau(r) \neq \tau(r')$. Thus, by letting $d = \tau(r)$ for some $r \in E$ (its choice does not matter), we have that ρt is a partial run of A' over wd. Overall, A' is locally concretisable, i.e. property (1) holds.

The last step concerns property (4). Intuitively, the idea is that if the data read corresponds to a data stored in some register, then the assignment can be replaced by

keeping in memory a relation between registers. This idea is merely an adaptation of the conversion from register automata ("M-automata", in their terminology) to finite-memory automata [KF94]. The states can be enriched with the right information to deal with these additional relations. □

In order to solve the unbounded register synthesis problem, we resort to a synthesis problem for data-free specifications. In that framework, when specifications are described by means of parity automata, synthesis problems can be solved using reductions to parity games. We thus quickly recall the notion of parity game. For a complete presentation, we refer the reader to [AG11].

A two-player parity game is given as a finite graph, in which vertices are partitioned among the two players, together with an initial vertex. A colouring function associates with each vertex an integer. It is used to define the winning plays as follows: a play is winning iff the maximum colour appearing infinitely often is even.

In the sequel, we will use the parity game associated with a DRA A, which is denoted as G_A. It is is defined as follows: its set of vertices is exactly that of A. Player Adam owns input vertices, and the associated input transitions, while player Eve owns output vertices/transitions. The colouring function is that of A, and the initial vertex is the initial state of A.

Proposition 3.6. *Let A be a $\mathsf{DRA}_{\mathsf{ido}}$ in good form. Then, the following are equivalent:*

(1) *$L(A)$ is realisable by a register transducer with as many registers as A*
(2) *$L(A)$ is realisable by an implementation*[2] *$I : (\Sigma_{\mathbb{i}} \times \mathcal{D})^* \to \Sigma_{\mathbb{o}} \times \mathcal{D}$*
(3) *Eve wins the parity game G_A associated with A*

Proof. We start with a preliminary remark on $\mathsf{DRA}_{\mathsf{ido}}$. As A is a $\mathsf{DRA}_{\mathsf{ido}}$, every output transition has a test with at least one equality constraint ($r^=$ for some r), and thus, as A is in good form (property (4)), the assignment of output transitions are all empty. Note that 1 $\Rightarrow$ 2 is immediate.

From the parity game G_A to the realisability of $L(A)$: 3 $\Rightarrow$ 1. Assume Eve wins the game G_A. Parity games admit memoryless strategies, i.e. strategies whose actions only depend on the current state of the game. We can thus consider a memoryless winning strategy for Eve, which we denote by a mapping χ from output vertices to output edges of the game, i.e. from output states to output transitions of A.

We now detail how we define from χ a register transducer T_χ with R^A as set of registers:

- States are those of A
- The initial state is that of A
- Transitions are defined as follows. Consider some input state p and some transition $t_{\mathbb{i}}$ from p to q. By definition of A, q is an output state, and we let $t_{\mathbb{o}} = \chi(q)$ be the transition given by Eve's strategy.

  We write $t_{\mathbb{i}} = (p, \sigma, \phi, \mathsf{asgn}, q)$ and $t_{\mathbb{o}} = (q, \sigma', \phi', \mathsf{asgn}', q')$. Thanks to our initial comment on the form of output transitions of $\mathsf{DRA}_{\mathsf{ido}}$ in good form, there exists a register r appearing with an equality constraint in the test ϕ' of the transition $t_{\mathbb{o}}$, and we have $\mathsf{asgn}' = \varnothing$. Then, we add to T_χ the transition $p \xrightarrow{\sigma,\phi|\mathsf{asgn},\sigma',r} q'$.

[2]Recall that implementations are defined in subsection 2.1.

Observe that T is indeed a register transducer as for each state p, it only uses transitions outgoing from p in A, hence it is deterministic as A was.

We claim that T_χ realises $L(A)$. Consider some input data word, and the behaviour of T_χ on this data word. As A is in good form, it is complete on its input states. This entails that this run is infinite. It corresponds to a play in G_A compatible with Eve's strategy χ. As χ is a winning strategy, this implies that the run is accepting, hence corresponds to some accepting run of A, yielding the result.

From the realisability of $L(A)$ to the parity game G_A: 2 $\Rightarrow$ 3. Assume that $L(A)$ is realisable by an implementation $I : (\Sigma_{\mathbb{I}} \times \mathcal{D})^* \to \Sigma_{\mathbb{O}} \times \mathcal{D}$. We let $f_I : \mathsf{DW}(\Sigma_{\mathbb{I}}, \mathcal{D}) \to \mathsf{DW}(\Sigma_{\mathbb{O}}, \mathcal{D})$ be the function it implements, and naturally extend it to finite words: for $w_{\mathbb{I}} \in \mathsf{DW}_f(\Sigma_{\mathbb{I}}, \mathcal{D}), f_I(w_{\mathbb{I}}) = I(w_{\mathbb{I}}[1])I(w_{\mathbb{I}}[1:2])\dots I(w_{\mathbb{I}}[1:|w_{\mathbb{I}}|])$. Let us build from I a winning strategy χ_I in G_A, with memory $(\Sigma_{\mathbb{I}} \times \mathcal{D})^* \times (Q_S \times \mathcal{D}^{R_A})$.

We define χ_I by induction, and show that when χ_I is in memory state $(w_{\mathbb{I}}, (q, \tau))$, the finite sequence of transitions constructed so far is a partial run of A over $\langle w_{\mathbb{I}}, f_I(w_{\mathbb{I}})\rangle$ ending in configuration (q, τ). Initially, χ_I has memory $(\varepsilon, (q_0, \tau_0))$.

Now, assume χ_I is in state $(w_{\mathbb{I}}, (q, \tau))$, and Adam just played $(\sigma_{\mathbb{I}}, \phi, \mathsf{asgn})$. Then, Eve picks some data $d_{\mathbb{I}} \in \mathcal{D}$ such that $\tau, d_{\mathbb{I}} \models \phi$. Such data exists since A is locally concretisable and the finite sequence of transitions constructed so far is the partial run over some data word. Let (q'', τ'') be the successor configuration of (q, τ) in A on reading $d_{\mathbb{I}}$, i.e. $(q, \tau) \xrightarrow[A]{\sigma_{\mathbb{I}}, d_{\mathbb{I}}} (q'', \tau'')$, and let $w'_{\mathbb{I}} = w_{\mathbb{I}}(\sigma_{\mathbb{I}}, d_{\mathbb{I}})$. Now, let $(\sigma_{\mathbb{O}}, d_{\mathbb{O}}) = I(w'_{\mathbb{I}})$. Correspondingly, let $t_{\mathbb{O}}$ be the transition taken from (q'', τ'') on reading $(\sigma_{\mathbb{O}}, d_{\mathbb{O}})$, i.e. such that $(q'', \tau'') \xrightarrow[t_{\mathbb{O}}]{\sigma_{\mathbb{O}}, d_{\mathbb{O}}} (q', \tau')$. Such transition exists: let $w \in \mathsf{DW}(\Sigma_{\mathbb{I}}, \mathcal{D})$ be some infinite suffix that we append to $w'_{\mathbb{I}}$. Since I is an implementation, f_I is total and we know that $\langle w'_{\mathbb{I}} w, f_I(w'_{\mathbb{I}} w)\rangle \in L(A)$, which means that $\langle w'_{\mathbb{I}} w, f_I(w'_{\mathbb{I}} w)\rangle$ admits an accepting run in A. In particular, its prefix $\langle w'_{\mathbb{I}}, f_I(w'_{\mathbb{I}})\rangle$ admits a partial run in A, and its last transition is $t_{\mathbb{O}}$ (such partial run is unique since A is deterministic).

Then, Eve plays $t_{\mathbb{O}}$ in G_A and updates her memory to $(w'_{\mathbb{I}}, (q', \tau'))$. The invariant indeed holds, as the play constructed so far is a partial run of A over $\langle w'_{\mathbb{I}}, f_I(w'_{\mathbb{I}})\rangle$ ending in configuration (q', τ').

χ_I is indeed a strategy, as it is defined for any possible sequence of actions of Adam. It remains to show that it is winning. Let ρ be a play consistent with χ_I, which is also a run of A by definition of G_A. We need to show that ρ is accepting. We define $w \in (\Sigma_{\mathbb{I}} \times \mathcal{D})^\omega$ as $w[i] = w^i_{\mathbb{I}}[i]$, where $w^i_{\mathbb{I}}$ is the input word stored in memory at step i of the play (i.e. such that χ_I is in state $(w^i_{\mathbb{I}}, (q_i, \tau_i))$ for some (q_i, τ_i) after receiving i actions of Adam). We then know that for all $i \in \mathbb{N}$, $\rho[:i]$ is a partial run of A over $\langle w[:i], f_I(w[:i])\rangle$, so ρ is a run of A over $\langle w, f_I(w)\rangle$. Since I is an implementation, such run is accepting, i.e. satisfies the parity condition, which means that ρ also satisfies the parity condition; it is thus winning. As a consequence, χ_I is a winning strategy in G_A. □

Theorem 3.7. Real($\mathsf{DRA}_{\mathsf{ido}}, \mathsf{RT}$) *is* ExpTime-c.

Proof. First, we put A in good form thanks to Lemma 3.5, resulting in some $\mathsf{DRA}_{\mathsf{ido}}$ B exponentially bigger. Then, by Proposition 3.6, it suffices to solve the parity game G_B. It is well-known to be possible in time $O(n^d)$ where n is the number of states and d the number of priorities. If n_A denotes the number of states of A and d its number of priorities, then B

has $n_A \cdot 2^{|R|^2}$ states and the same number of priorities d, hence checking the realisability of A can be done in time $O(n_A^d \cdot 2^{d \cdot |R|^2})$, which is exponential with respect to the size of the input.

Hardness. The following proof is an adaptation of the one establishing PSPACE-hardness of the nonemptiness problem for DRA presented in [DL09, Theorem 5.1]. Here, we use the input part to simulate universal transitions, and the output part to simulate nondeterministic ones, hence simulating alternation, which yields an EXPTIME lower bound.

Thus, we reduce from the halting problem of alternating Turing machines over a binary alphabet with a linearly bounded tape. An alternating Turing machine is a tuple $\mathcal{M} = \langle Q, q_i, \delta \rangle$, where:

- Q is a finite set of *states*, partitioned into *existential* ($Q_\exists$) and *universal* ($Q_\forall$) states: $Q = Q_\exists \uplus Q_\forall$, where $q_i \in Q_\forall$ is the *initial* state
- $\delta : Q \times \{0,1\} \to 2^{Q \times \{0,1\} \times \{-1,1\}}$ is the *transition function.*

A configuration of $\mathcal{M}$ is then a triple $c = (q, i, w)$, where $q \in Q$ is the machine state, $i \in \{0, \dots, |\mathcal{M}| - 1\}$ is the head position, and $w \in \{0,1\}^{|\mathcal{M}|}$ is the tape content. It is *existential* if $q \in Q_\exists$ and *universal* if $q \in Q_\forall$. A configuration (q', i', w') is a successor of (q, i, w) if there exists $(p, a, m) \in \delta(q, w[i])$, $p = q'$, $i' = i + m \in \{0, \dots, |\mathcal{M}| - 1\}$ and w' is such that $\forall j \neq i$, $w'[j] = w[j]$ and $w[i] = a$. $t = q \xrightarrow{w[i],a,m} p$ is called the *associated transition.* A *run* of $\mathcal{M}$ is then a tree whose nodes are configurations and whose branches can be finite or infinite, rooted in the initial configuration $(q_i, 0, 0^{|\mathcal{M}|})$, and whose nodes satisfy the following properties:

(1) If the node is an existential configuration $c_\exists$, then it has exactly one child, which is a successor configuration of $c_\exists$.
(2) If the node is a universal configuration $c_\forall$, then its children are all its successor configurations.

Note that a branch is finite if and only if it ends in a universal configuration with no successor. The machine $\mathcal{M}$ *halts* if it admits a run which is a finite tree (i.e. whose branches all end in a universal configuration with no successors). The following problem is EXPTIME-hard [CKS81]: given an alternating Turing machine $\mathcal{M}$, decide whether $\mathcal{M}$ halts.

Finally, a computation is a finite sequence of successive configurations (i.e. a finite path in a run). Let $(q_0, i_0, w_0) \dots (q_n, i_n, w_n)$ be a computation of $\mathcal{M}$, and $t_0 \dots t_{n-1}$ the sequence of associated transitions. We encode such computation by the following data word over the alphabet $Q \uplus \delta \uplus \{-\}$:

$$(-, d_0)(-, d_1) a_0^0 a_1^0 \dots a_{|\mathcal{M}|-1}^0 t_0 a_0^1 a_1^1 \dots a_{|\mathcal{M}|-1}^1 t_1 \dots t_{n-1} a_0^n a_1^n \dots a_{|\mathcal{M}|-1}^n$$

where $d_0 \neq d_1 \in \mathcal{D}$ are two distinct data respectively encoding letters 0 and 1, and we have $\mathsf{lab}(a_l^k) = q_k$ if $l = i_k$ and $\mathsf{lab}(a_l^k) = -$ otherwise. Then, $\mathsf{dt}(a_l^k) = d_0$ if $w_k[l] = 0$ and $\mathsf{dt}(a_l^k) = d_1$ if $w_k[l] = 1$. $\mathsf{dt}(t_k)$ does not matter.

Now, as in [DL09], we can construct a DRA $A_\mathcal{M}$ which accepts a data word iff it has a prefix that encodes a computation of $\mathcal{M}$ from the initial state to a state with no successor. Indeed, the transitions are part of the input, so they do not have to be guessed: neither nondeterministic nor universal branching is needed here (they will respectively be simulated by the output and input player). For completeness, we describe the construction: $A_\mathcal{M}$ has memory Q, along with an $|\mathcal{M}|$-bounded counter l to keep track of the position of the reading

head in w_k, a variable i taking its values in $\{0, \dots, |\mathcal{M}|-1\}$ used to store the value of i_k and a variable t taking its values in δ to memorise t_k; which overall yields a $O(|\mathcal{M}|^4)$ memory. Its finite alphabet is $Q \uplus \delta \uplus \{-\}$, and it has $|\mathcal{M}|+2$ registers: r_0 and r_1 respectively store d_0 and d_1, and, for all $0 \leq l < |\mathcal{M}|$, r'_l successively stores the different values of $w_k[l]$ for $0 \leq k \leq n$. Then, a run of $A_{\mathcal{M}}$ is as follows: initially, $A_{\mathcal{M}}$ stores d_0 and d_1, while checking that they are distinct. Then, it checks that $w_0 = 0^{|\mathcal{M}|}$. To check successorship, while maintaining the invariant that at any step k, r'_l contains $w_k[l]$, the automaton, when reading $t_k = q \xrightarrow{c,a,m} p$, checks that $q = q_k$ (it was stored as the target of t_{k-1}), $c = w_k[i_k]$ (i.e. that r'_{i_k} contains d_c), and updates the value of i_k to $i_{k+1} = i_k + m_k$, while checking that $i_k \in \{0, \dots, |\mathcal{M}|-1\}$. Then, with the help of its registers and its counter l, it checks that $w_{k+1}[l] = w_k[l]$ for all $l \neq i_{k+1}$, and that $w_{k+1}[i_{k+1}] = d_a$.

From such automaton, by adding #s to enforce the alternation between input and output, we can build a specification automaton such that the input player provides the encoding of the successive configurations, and resolves the universal branching, and the output player has to resolve nondeterminism (i.e. chooses which nondeterministic transition to take). Then, if the input player can force the computation to go on ad infinitum, he wins, otherwise (if either the provided encoding is not correct, or if the computation is finite), the output player wins. Formally:

$$\begin{aligned} S = &\{(-,d_0)\#(-,d_1)\#\langle c_0, \#^{|\mathcal{M}|}\rangle t_0 \# \langle c_1, \#^{|\mathcal{M}|}\rangle \# t_1 \langle c_2, \#^{|\mathcal{M}|}\rangle t_2 \# \dots \langle c_n, \#^{|\mathcal{M}|}\rangle \#^\omega \mid \\ &\quad d_0 \neq d_1 \text{ and } c_0 t_0 c_1 t_1 c_2 t_2 \dots t_{n-1} c_n \text{ is the encoding of a computation of } \mathcal{M}\} \\ &\cup \left\{ \langle w, w' \rangle \;\middle|\; \begin{array}{c} \text{there exists a prefix of } w \text{ which is not} \\ \text{the encoding of a computation of } \mathcal{M} \end{array} \right\} \\ &\cup \{\langle (-,d_0)\#(-,d_1)w, w' \rangle \mid d_0 = d_1\} \end{aligned}$$

The data corresponding to the # and t_i do not matter, and are not depicted. Note that the even (i.e. universal) transitions are picked by the input player, while the odd (i.e. nondeterministic) transitions are picked by the output player.

Now, if $\mathcal{M}$ halts, A admits an implementation, which behaves as follows: it first checks that the d_0 and d_1 given as input are indeed distinct. Then, it checks on-the-fly that the given input is indeed an encoding of the initial configuration, while outputting #s. It then checks that c_1 is indeed a successor of c_0 following t_0, again while outputting #s. Then, if it receives a # as input, it picks some t_1 which is a witness that c_0 is indeed accepting, and so on. If, at some point, the given input is not a valid encoding, then it behaves arbitrarily (e.g. by outputting only #s).

Conversely, if $\mathcal{M}$ does not halt, then, by choosing an input whose universal transitions are witnesses that c_0 is not accepting, then either the implementation provides some non-admissible output at some point, or the computation goes ad infinitum, which breaks the specification.

For readers familiar with game-theoretic formulations, winning strategies in the synthesis game of $A_{\mathcal{M}}$ are in one-to-one correspondence with halting runs of $\mathcal{M}$. □

As a consequence of the fact that if a $\mathsf{DRA}_{\mathsf{ido}}$ is realisable, then it is so by a register transducer with the same number of registers, we obtain the following corollary:

Corollary 3.8. *Let $k \geq r$ be two integers. We denote by $\mathsf{DRA}_{\mathsf{ido}}[r]$ the class of $\mathsf{DRA}_{\mathsf{ido}}$ with r registers.* $\textsc{Real}(\mathsf{DRA}_{\mathsf{ido}}[r], \mathsf{RT}[k])$ *is in* $\textsc{ExpTime}$.

4. Bounded Synthesis: A Generic Approach

In this section, we study the setting where target implementations are register transducers in the class $\mathsf{RT}[k]$, for some $k \geq 0$ that we now fix for the whole section. For the complexity analysis, we assume k is given as input, in unary. Indeed, describing a k-register automaton in general requires $O(k)$ bits, and not $O(\log k)$ bits. We prove the decidable cases of the first line of Table 1 (page 4), by reducing the problems to realisability problems for data-free specifications.

4.1. **Abstract Actions.** We let $R_k = \{1, \dots, k\}$ be a set of k registers. Our aim is to reduce the problem to a finite alphabet problem. First, since the set of test formulas over R_k is infinite and there are doubly exponentially many non-equivalent formulas over R_k, we rather synthesise transducers whose tests are maximally consistent conjunctions of atoms of the form $r^=$ or $r^\neq$. Such conjunctions can be identified as subsets of R_k in a natural way, e.g. for $k = 3$, the test $r_1^= \wedge r_2^\neq \wedge r_3^=$ is identified with the set $\{1, 3\}$. We call them explicit tests and denote them by the capital letter E. An explicit test $E \subseteq R_k$ is converted into the (implicit) test $\phi_E = \bigwedge_{r \in E} r^= \wedge \bigwedge_{r \notin E} r^\neq$. Explicit tests are for instance used in [Seg06].

We let $\mathsf{Tst}_k = \mathsf{Asgn}_k = 2^{R_k}$. The finite input actions are $A_{\mathbb{I}}^k = \Sigma_{\mathbb{I}} \times \mathsf{Tst}_k$ which corresponds to picking a label and a test over the k registers, and the output actions are $A_{\mathbb{O}}^k = \Sigma_{\mathbb{O}} \times \mathsf{Asgn}_k \times R_k$, corresponding to picking some output symbol, some assignment and some register whose content is to be output.

An alternating sequence of actions $\overline{a} = (\sigma_{\mathbb{I}}^1, E_1)(\sigma_{\mathbb{O}}^1, \mathsf{asgn}_1, r_1) \cdots \in (A_{\mathbb{I}}^k A_{\mathbb{O}}^k)^\omega$ abstracts a set of relational data words of the form $w = (\sigma_{\mathbb{I}}^1, d_{\mathbb{I}}^1)(\sigma_{\mathbb{O}}^1, d_{\mathbb{O}}^1) \cdots \in \mathsf{RW}(\Sigma_{\mathbb{I}}, \Sigma_{\mathbb{O}}, \mathcal{D})$ via a compatibility relation that we now define. We say that w is *compatible* with $\overline{a}$ if there exists a sequence of register configurations $\tau_0 \tau_1 \cdots \in (R_k \to \mathcal{D})^\omega$ such that $\tau_0 = \tau_0^{R_k}$ and for all $i \geq 1$, $\tau_i, d_{\mathbb{I}}^i \models E_i$, $d_{\mathbb{O}}^i = \tau_i(r_i)$ and $\tau_{i+1} = \mathrm{next}(\tau_i, d_{\mathbb{I}}^i, \mathsf{asgn}_i)$. In other words, w is *compatible* with $\overline{a}$ if there exists some k-register transducer and a run $\rho = t_0 t_1 \dots$ such that for all i, t_i is of the form $t_i = q_i \xrightarrow{\sigma_{\mathbb{I}}^i, E_i | \sigma_{\mathbb{O}}^i, \mathsf{asgn}_i, r_i} q_{i+1}$ for some $q_i, q_{i+1} \in Q_T$. Note that this sequence is unique if it exists. We denote by $\mathsf{Comp}(\overline{a})$ the set of relational data words compatible with $\overline{a}$. Given a specification S, we let $W_{S,k} = \{\overline{a} \mid \mathsf{Comp}(\overline{a}) \subseteq S\}$. The set $W_{S,k}$ is then a specification over the finite input and output alphabets $A_{\mathbb{I}}^k$ and $A_{\mathbb{O}}^k$.

Theorem 4.1 (Transfer)**.** *Let S be a data word specification. The following are equivalent:*
(1) *S is realisable by a transducer with k registers.*
(2) *The (data-free) word specification $W_{S,k}$ is realisable by a (register-free) finite transducer.*

Proof. Let T be a transducer with k registers realising S. The tests of T are implicit tests, so in a first step we explicit them, possibly by adding new transitions to T. Formally, a transition $q \xrightarrow[T]{\sigma_{\mathbb{I}}, \phi | \sigma_{\mathbb{O}}, \mathsf{asgn}, r} q'$ is replaced by all the transitions $q \xrightarrow[T]{\sigma_{\mathbb{I}}, E | \sigma_{\mathbb{O}}, \mathsf{asgn}, r} q'$ for all $E \subseteq R_k$ such that $\phi_E \Rightarrow \phi$ is true. The resulting transducer can be seen as a finite transducer T' over input alphabet $A_{\mathbb{I}}^k$ and output alphabet $A_{\mathbb{O}}^k$. Moreover, since the transition function of T is complete, it is also the case of T' (this is required by the definition of transducer defining implementations).

Let us show that $W_{S,k}$ is realisable by T', i.e. $L(T') \subseteq W_{S,k}$. Take a sequence $\overline{a} = a_1 e_1 a_2 e_2 \cdots \in L(T')$. We show that $\mathsf{Comp}(\overline{a}) \subseteq S$. Let $w \in \mathsf{Comp}(\overline{a})$. Then, there exists a run $q_0 q_1 q_2 \dots$ of T' on $\overline{a}$ since $\overline{a} \in L(T')$. By definition of compatibility for w, there exists a sequence of register configurations $\tau_0 \tau_1 \cdots \in (R_k \to \mathcal{D})^\omega$ satisfying the conditions in the

definition of compatibility. From this we can deduce that $(q_0, \tau_0)(q_1, \tau_1)\dots$ is an initial sequence of configurations of T over w, so $w \in L(T)$. Finally, T realises S, and therefore $L(T) \subseteq S$.

Conversely, suppose that $W_{S,k}$ is realisable by some finite transducer T' over the input (output) alphabets $A_{\mathbb{I}}^k$ ($A_{\mathbb{O}}^k$). Again, the transducer T' can be seen as a transducer T with k registers over data words with explicit tests. We show that T realises S, i.e., $L(T) \subseteq S$. Let $w \in L(T)$. The run of T over w induces a sequence of actions $\overline{a}$ in $(A_{\mathbb{I}}^k A_{\mathbb{O}}^k)^\omega$ which, by definition of compatibility, satisfies $w \in \mathsf{Comp}(\overline{a})$. Moreover, $\overline{a} \in L(T')$. Hence, since T' realises $W_{S,k}$, we get $\mathsf{Comp}(\overline{a}) \subseteq S$, so $w \in S$, concluding the proof. □

4.2. **The case of URA specifications.** In this section, we show that for any S a data word specification given as some URA, the language $W_{S,k}$ is effectively ω-regular, entailing the decidability of Real(URA, RT[k]), by Theorem 4.1 and the decidability of (data-free) synthesis. Let us first prove a series of intermediate lemmas.

We define an operation $\otimes$ between relational data words $w \in \mathsf{RW}(\Sigma_{\mathbb{I}}, \Sigma_{\mathbb{O}}, \mathcal{D})$ and sequences of actions $\overline{a} \in (A_{\mathbb{I}}^k A_{\mathbb{O}}^k)^\omega$ as follows: $w \otimes \overline{a} \in \mathsf{RW}(A_{\mathbb{I}}^k, A_{\mathbb{O}}^k, \mathcal{D})$ is defined only if for all $i \geq 1$, $\mathsf{lab}(w[i]) = \mathsf{lab}(\overline{a}[i])$ where $\mathsf{lab}(\overline{a}[i])$ is the first component of $\overline{a}[i]$ (a label in $\Sigma_{\mathbb{I}} \cup \Sigma_{\mathbb{O}}$), by $(w \otimes \overline{a})[i] = (\overline{a}[i], \mathsf{dt}(w[i]))$. Note that such operation is always defined when $w \in \mathsf{Comp}(\overline{a})$.

Lemma 4.2. *The language $L_k = \{w \otimes \overline{a} \mid w \in \mathsf{Comp}(\overline{a})\}$ is definable by some* NRA.

Proof. We define an NRA with k registers which roughly follows the actions it reads on its input. Its set of states is $\{q\} \cup \mathsf{Asgn}_R$, with initial state q. In state q, it is only allowed to read labelled data in $A_{\mathbb{I}}^k \times \mathcal{D}$. On reading $(\sigma_{\mathbb{I}}, \phi, d)$, it guesses some assignment asgn, performs the test ϕ and the assignment asgn and goes to state asgn. In any state $\mathsf{asgn} \in \mathsf{Asgn}_R$, it is only allowed to read labelled data of the form $(\sigma_{\mathbb{O}}, \mathsf{asgn}, r, d)$, for which it tests whether d is equal to the content of r. It does no assignment and moves back to state q. All states are accepting (i.e. have parity 0). Such NRA has size $O(2^{k^2})$. □

Let S be a specification defined by some URA A_S with set of states Q. The following subset of L_k is definable by some NRA, where $\overline{S}$ denotes the complement of S:

Lemma 4.3. *The language $L_{\overline{S},k} = \{w \otimes \overline{a} \mid w \in \mathsf{Comp}(\overline{a}) \cap \overline{S}\}$ is definable by some* NRA.

Proof. Since S is definable by the URA A_S, $\overline{S}$ is NRA-definable with $\overline{A_S}$, a copy of A_S with colouring function $\overline{c} : q \mapsto c(q) + 1$, interpreted as an NRA. Let B be some NRA defining L_k (it exists by Lemma 4.2). It now suffices to take a product of $A_{\overline{S}}$ and B to get an NRA defining $L_{\overline{S},k}$. □

Given a data word language L, we denote by $\mathsf{lab}(L) = \{\mathsf{lab}(w) \mid w \in L\}$ its projection on labels. The language $W_{S,k}$ is obtained as the complement of the label projection of $L_{\overline{S},k}$:

Lemma 4.4. $W_{S,k} = \overline{\mathsf{lab}(L_{\overline{S},k})}$.

Proof. Let $\overline{a} \in (A_{\mathbb{I}}^k A_{\mathbb{O}}^k)^\omega$. Then, $\overline{a} \notin W_{S,k} \Leftrightarrow \mathsf{Comp}(\overline{a}) \not\subseteq S \Leftrightarrow \exists w \in \mathsf{RW}, w \in \mathsf{Comp}(\overline{a}) \cap \overline{S} \Leftrightarrow \exists w \in \mathsf{RW}, w \otimes \overline{a} \in L_{\overline{S},k} \Leftrightarrow \overline{a} \in \mathsf{lab}(L_{\overline{S},k})$. □

We are now able to show the regularity of $W_{S,k}$.

Lemma 4.5. *Let S be a data word specification, $k \geq 0$. If S is definable by some* URA *with n states and r registers, then $W_{S,k}$ is effectively ω-regular, definable by some deterministic parity automaton with $O(2^{n^2 \cdot 16^{(r+k)^2}})$ states and $O(n \cdot 4^{(r+k)^2})$ priorities.*

Proof. First, $L_{\overline{S},k}$ is definable by some NRA with $O(2^{k^2} n)$ states and $O(r+k)$ registers by Lemma 4.4, obtained as product between the NRA $\overline{A_S}$ and the automaton obtained in Lemma 4.2, of size $O(2^{k^2})$. It is known that the projection on the alphabet of labels of a language of data words recognised by some NRA is effectively regular [KF94]. The same construction, which is based on extending the state space with register equality types, carries over to ω-words, and one obtains a nondeterministic parity automaton with $O(n \cdot 4^{(r+k)^2})$ states and d priorities recognising $\mathsf{lab}(L_{\overline{S},k})$. It can be complemented into a deterministic parity automaton with $O(2^{n^2 \cdot 16^{(r+k)^2}})$ states and $O(n \cdot 4^{(r+k)^2})$ priorities using standard constructions [Pit07]. □

We are now able to reprove the following result, known from [KMB18]:

Theorem 4.6. *For all $k \geq 0$, $\textsc{Real}(\mathsf{URA}, \mathsf{RT}[k])$ is in $2\textsc{ExpTime}$.*

Proof. By Lemma 4.5, we construct a deterministic parity automaton $P_{S,k}$ for $W_{S,k}$. Then, according to Theorem 4.1, it suffices to check whether it is realisable by a (register-free) transducer. The way to decide it is to see $P_{S,k}$ as a two-player parity game and check whether the protagonist has a winning strategy. Parity games can be solved in time $O(m^{\log d})$ [CJK+17] where m is the number of states of the game and d the number of priorities. Overall, solving it requires doubly exponential time, more precisely in $O(2^{n^3 \cdot 16^{(r+k)^2}})$. □

4.3. **The case of test-free NRA specifications.** Unfortunately, by Theorem 3.2, the synthesis problem for specifications expressed as NRA is undecidable, even when the number of registers of the implementation is bounded. And indeed, if we mimic the reasoning of the previous section, we get that $L_{\overline{S},k}$ is definable by a URA, but Lemma 4.4 does not allow to conclude because:

Proposition 4.7. *There exists a data word language L which is* URA*-definable and whose string projection is not ω-regular.*

Proof. Consider

$$L = \{(r, d_1) \dots (r, d_n)(g, d'_1) \dots (g, d'_m)(\#, d)^\omega \mid \forall i \neq j, d_i \neq d_j \wedge \forall 1 \leq i \leq n, \exists j, d'_j = d_i\},$$

which consists in a word $w \in r^n$ with pairwise distinct data followed by a word $w' \in g^m$ which contains at least all the data of w, and extended with $(\#, d)^\omega$ to make it infinite (here, the choice of d does not matter). Such language can be interpreted as the request-grant specification, restricted to the case where all requests are made first, and are all made by pairwise distinct clients (plus a # infinite padding). L is recognised by an URA which, on reading (r, d_i), universally triggers a run checking that

(1) Once a label g is read, only gs are read; and after the last g, only # are read (this is an ω-regular property)
(2) (r, d_i) does not appear again
(3) (g, d_i) appears at least once.

Now, we have $\mathsf{lab}(L) = \{r^n g^m \#^\omega \mid m \geq n\}$, which is not ω-regular. □

In this section, we consider the class of NRA which do not perform tests on input data, which we call test-free nondeterministic register automata ($\mathsf{NRA_{tf}}$ for short). Such restriction is inspired from [DH16], which defines transformations of data words using MSO interpretations with an MSO origin relation. The MSO interpretation describes the transformation over the finite alphabet (called the *string transduction*), as in [Cou94], while the MSO origin relation describes the relation between input and output data. Such relation does not depend on (un)equalities between different input data: it uniquely maps each output position to an input position, expressing that the output data at this position is equal to the corresponding input data. They then show that such model is equivalent to two-way deterministic transducers with *data variables*[3]. Such data variables are used to implement the MSO origin relation: they are registers in which the transducer can store the input data values and output them, but it is not allowed to perform any test on the stored data, contrary to our model of register automata. To define $\mathsf{NRA_{tf}}$, we apply the same restriction to NRA: they correspond to *nondeterministic one-way* transducers with data variables. Such machines can only rearrange input data (duplicate, erase, copy) regardless of the actual data values (as there are no tests). This way, as stated in Proposition 4.9, registers induce an origin relation between input and output data.

To avoid confusion between the nature of specifications and implementations, we prefer to define them as register automata, instead of transducers.

Definition 4.8 (Test-free register automaton)**.** A NRA is *test-free* if:

(1) Its input transitions do not depend on equality relations between input data: for all $t \in \delta$, if $t = q \xrightarrow{\sigma,\phi,\mathsf{asgn}} q'$ is an input transition, then $\phi = \top$.
(2) Its output transitions consist in outputting the content of some register: for all $t \in \delta$, if $t = q \xrightarrow{\sigma,\phi,\mathsf{asgn}} q'$ is an output transition, then $\phi = r^=$ for some $r \in R$ and $\mathsf{asgn} = \varnothing$.

We now make the relation with the notion of origin precise: as shown in [DFL18], there is a tight connection between origin graphs and data words. Here, the encoding is slightly different, as we do not necessarily ask that the data labelling input position n is equal to n. However, as long as the input data are all pairwise distinct, such encoding carries to our setting: the output data at position j is equal to $d^i_{\mathbb{I}}$, where i is the (input) origin position. Thus, in the following, we let AllDiff denote the set of relational data words whose input data are pairwise distinct:

$$\textsc{AllDiff} = \{w = (\sigma^1_{\mathbb{I}}, d^1_{\mathbb{I}})(\sigma^1_{\mathbb{O}}, d^1_{\mathbb{O}}) \cdots \in \mathsf{RW} \mid \forall 0 \leq i < i', d^i_{\mathbb{I}} \neq d^{i'}_{\mathbb{I}}\}$$

where, by convention $d^0_{\mathbb{I}} = \mathsf{d}_0$. Then, as we will show, the behaviour of an $\mathsf{NRA_{tf}}$ over AllDiff determines its origin relation, and hence its behaviour over the entire data domain.

To a run $\rho = q_0 \xrightarrow{\sigma^1_{\mathbb{I}},\mathsf{asgn}^1,r^1,\sigma^1_{\mathbb{O}}} q_1 \xrightarrow{\sigma^2_{\mathbb{I}},\mathsf{asgn}^2,r^2,\sigma^2_{\mathbb{O}}} q_2 \ldots$, we associate the origin function $o_\rho : j \mapsto \max\{i \leq j \mid r_j \in \mathsf{asgn}_i\}$, with the convention $\max \varnothing = 0$. In other words, $o_\rho(j)$ is the last input position at which the register output at position j was assigned, so the corresponding input data is the one which is output (if the register has never been assigned, it contains d_0, which, by convention, is the data associated with input position 0).

Now, for an origin function $o : \mathbb{N}\backslash\{0\} \to \mathbb{N}$ and for a relational data word $w \in \mathsf{RW}$, we say w is compatible with the origin function o, denoted $w \models o$, whenever for all $j \geq 1$, $\mathsf{dt}(\mathsf{out}(w)[j]) = \mathsf{dt}(\mathsf{inp}(w)[o(j)])$, with the convention $\mathsf{dt}(\mathsf{inp}(w)[0]) = \mathsf{d}_0$.

[3]Themselves equivalent to one-way streaming string transducers with data variables and parameters; such parameters are reminiscent of the guessing mechanism described in [KZ10].

The following proposition shows that actual data values in a word w do not matter with respect to membership in some $\mathsf{NRA}_{\mathsf{tf}}$, only the compatibility with origin functions does:

Proposition 4.9. *Let $w \in RW$ and ρ a sequence of transitions of some $\mathsf{NRA}_{\mathsf{tf}}$. Then,*
(1) *If ρ is a run over w, then $w \models o_\rho$.*
(2) *If ρ is a run over w and $w \in$ AllDiff, then for all $o : \mathbb{N}\backslash\{0\} \to \mathbb{N}$, $w \models o \Leftrightarrow o = o_\rho$.*
(3) *If w and ρ have the same finite labels and if $w \models o_\rho$, then ρ is a run over w.*

Proof. (1) and (3) follow from the semantics of $\mathsf{NRA}_{\mathsf{tf}}$, which do not conduct any test on the input data. The $\Leftarrow$ direction of (2) is exactly (1). Now, assume $w \in$ AllDiff admits ρ as a run, and let o such that $w \models o$. Then, let $j \geq 1$ be such that $\mathsf{dt}(\mathsf{out}(w)[j]) = \mathsf{dt}(\mathsf{inp}(w)[o(j)])$. By (1) we know that $\mathsf{dt}(\mathsf{out}(w)[j]) = \mathsf{dt}(\mathsf{inp}(w)[o_\rho(j)])$, so $\mathsf{dt}(\mathsf{inp}(w)[o(j)]) = \mathsf{dt}(\mathsf{inp}(w)[o_\rho(j)])$. Since $w \in$ AllDiff, this implies $o(j) = o_\rho(j)$, so, overall, $o = o_\rho$. □

It is not clear whether $W_{S,k}$ is regular for $\mathsf{NRA}_{\mathsf{tf}}$ specifications, but we show that it suffices to consider another set denoted $W^{\mathsf{tf}}_{S,k}$ which is easier to analyse (and can be proven regular), which describes the behaviour of S over input with pairwise distinct data. Indeed, as expressed by the above proposition, $\mathsf{NRA}_{\mathsf{tf}}$ cannot conduct tests on input data, and their behaviour only depends on the input labels. Thus, it suffices to study runs on input words whose data are all distinct; such choice ensures that two equal input data will not ease the task of the implementation. Otherwise, it could be that on reading a data word, two registers r_1 and r_2 are equal, and then the implementation can simultaneously take transitions labelled with $\mathsf{out}(r_1)$ and $\mathsf{out}(r_2)$. An interesting side-product of this approach is that it implies that we can restrict to test-free implementations. A test-free transducer is a transducer whose transitions do not depend on tests over input data, i.e., for all transitions $t = q \xrightarrow{\sigma_{\mathbb{I}},\phi|\mathsf{asgn},\sigma_{\mathbb{O}},r} q' \in \delta$, we have $\phi = \top$.

Proposition 4.10. *Let S be a $\mathsf{NRA}_{\mathsf{tf}}$ specification, and $A^{\varnothing}_{\mathbb{I}} = \Sigma_{\mathbb{I}} \times \{\varnothing\}$. The following are equivalent:*
(1) *S is realisable*
(2) *$W^{\mathsf{tf}}_{S,k} = \{\overline{a} \in (A^{\varnothing}_{\mathbb{I}} A^k_{\mathbb{O}})^\omega \mid \mathsf{Comp}(\overline{a}) \cap S \cap$ AllDiff $\neq \varnothing\}$ is realisable by a (register-free) transducer with input alphabet $A^{\varnothing}_{\mathbb{I}}$*
(3) *S is realisable by a test-free transducer*

Proof. (3) $\Rightarrow$ (1) is trivial.

(1) $\Rightarrow$ (2): If S is realisable, then, by Theorem 4.1, $W_{S,k}$ is realisable by some transducer I. Now, since transducers are closed under regular domain restriction, $W^{\varnothing}_{S,k} = W_{S,k} \cap (A^{\varnothing}_{\mathbb{I}} A^k_{\mathbb{O}})^\omega$ is realisable by I restricted to the input alphabet $A^{\varnothing}_{\mathbb{I}}$; more precisely, by the transducer I' with the same set of states as I and transition function $\delta' = \delta \cap \big(Q_I \times \Sigma_{\mathbb{I}} \times \{\varnothing\} \to \mathsf{Asgn}_{R_k} \times \Sigma_{\mathbb{O}} \times R_k \times Q_I\big)$. Moreover, $W^{\varnothing}_{S,k} \subseteq W^{\mathsf{tf}}_{S,k}$. Indeed, let $\overline{a} \in W^{\varnothing}_{S,k}$. Then, $\mathsf{Comp}(\overline{a}) \subseteq S$. It is easy to build by induction a data word $w \in \mathsf{Comp}(\overline{a}) \cap$ AllDiff, so $\mathsf{Comp}(\overline{a}) \cap S \cap$ AllDiff $\neq \varnothing$. Thus, $W^{\mathsf{tf}}_{S,k}$ is realisable by any transducer realising $W^{\varnothing}_{S,k}$.

(2) $\Rightarrow$ (3): Now, assume $W^{\mathsf{tf}}_{S,k}$ is realisable by some transducer I. We show that I, when ignoring the $\varnothing$ input tests, is actually an implementation of S. Thus, let I' be the same transducer as I except that all input tests $\varnothing$ have been replaced with $\top$. Formally, $q \xrightarrow[I']{\sigma_{\mathbb{I}},\top|\mathsf{asgn},\sigma_{\mathbb{O}},r} q'$ iff $q \xrightarrow[I]{\sigma_{\mathbb{I}},\varnothing|\mathsf{asgn},\sigma_{\mathbb{O}},r} q'$ Note that I', interpreted as a register transducer, is test-free. Let $w \in \mathsf{DW}$, and $\overline{a}_{\mathbb{I}} = \mathsf{lab}(w) \times \varnothing^\omega$ be the input action in $A^{\varnothing}_{\mathbb{I}}$ with same finite

labels as w. Let $\overline{a} = I(\overline{a}_{\mathbb{i}})$, and let $w' \in \mathsf{Comp}(\overline{a}) \cap S \cap \textsc{AllDiff}$ (such w' exists because, as above, $\mathsf{Comp}(\overline{a}) \cap \textsc{AllDiff} \neq \varnothing$). Then, since $\mathsf{lab}(w) = \mathsf{lab}(w')$, they admit the same run ρ^I in I, so $w, w' \models o_{\rho^I}$. Now, $w' \in S$, so it admits an accepting run ρ^S in S, which implies $w' \models o_{\rho^S}$. Moreover, $w' \in \textsc{AllDiff}$ so, by Proposition 4.9 (2), we get $o_{\rho^I} = o_{\rho^S}$. Therefore, $w \models o_{\rho^S}$, so, by Proposition 4.9 (3), w admits ρ^S as a run, i.e. $w \in S$. Overall, $L(I) \subseteq S$, meaning that I is a (test-free) implementation of S. □

Finally, $W^{\mathsf{tf}}_{S,k} = \{\overline{a} \in (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega \mid \mathsf{Comp}(\overline{a}) \cap S \cap \textsc{AllDiff} \neq \varnothing\}$ is regular. Indeed, $W^{\mathsf{tf}}_{S,k} = \{\overline{a} \in (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega \mid \mathsf{Comp}(\overline{a}) \cap S^{\varnothing} \neq \varnothing\}$, where $S^{\varnothing}$ is the same automaton as S except that all input transitions $q \xrightarrow{\sigma_{\mathbb{i}}, \top, \mathsf{asgn}} q'$ have been replaced with $q \xrightarrow{\sigma_{\mathbb{i}}, \bigwedge_{r \in R_k} r^{\neq}, \mathsf{asgn}} q'$, because, for all $\overline{a} \in (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$, $\mathsf{Comp}(\overline{a}) \cap S \cap \textsc{AllDiff} \neq \varnothing \Leftrightarrow \mathsf{Comp}(\overline{a}) \cap S^{\varnothing} \neq \varnothing$ (the $\Rightarrow$ direction is trivial, and the $\Leftarrow$ stems from the fact that an $\textsc{AllDiff}$ input only takes $\phi = \varnothing$ transitions).

Then, $L^{\mathsf{tf}}_{S,k} = \{w \otimes \overline{a} \in \mathsf{RW} \otimes (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega \mid w \in \mathsf{Comp}(\overline{a}) \cap S^{\varnothing}\}$ is NRA-definable. Indeed, S is $\mathsf{NRA}_{\mathsf{tf}}$-definable, so $S^{\varnothing}$ is NRA-definable, and by Lemma 4.2, $L_k = \{w \otimes \overline{a} \mid w \in \mathsf{Comp}(\overline{a})\}$ is NRA-definable, so their product recognises $L^{\mathsf{tf}}_{S,k}$. Finally, $W^{\mathsf{tf}}_{S,k} = \mathsf{lab}(L^{\mathsf{tf}}_{S,k})$, and the projection of a NRA over some finite alphabet is regular [KF94].

Overall, by Theorem 4.1, we finally get (the complexity analysis is the same as for URA):

Theorem 4.11. *For all $k \geq 0$, $\textsc{Real}(\mathsf{NRA}_{\mathsf{tf}}, \mathsf{RT}[k])$ is decidable and in $2\textsc{ExpTime}$.*

5. Synthesis and Uniformisation

In this section, we discuss the connection between synthesis and uniformisation of relations, which is a more general problem: as pointed out in Section 2, if S is realisable by a register transducer, then, in particular, it has a total domain, i.e. $\mathsf{inp}(S) = \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$, otherwise it cannot be that $L(T) \subseteq S$ for T a register transducer, since by definition of transducers $\mathsf{inp}(T) = \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$. However, when defining a specification, the user might be interested only in a subset of behaviours (for instance, s/he knows that all input data will be pairwise distinct). In the finite alphabet setting, since the formalisms used to express specifications are closed under complement (whether it is LTL or ω-automata), it is actually not a restriction to assume that the input domain of the specification is total: it suffices to complete the specification by allowing any behaviour on the input not considered. However, since register automata are not closed under complement, such approach is not possible here. Thus, it is relevant to generalise the realisability problem to the case where the domain of the specification is not total. This can be done by equipping register transducers with an acceptance condition. It is also necessary to adapt the notion of realisability; otherwise, any transducer accepting no words realises any specification. (since it is always the case that $\varnothing \subseteq S$). A natural way is to consider synthesis as a uniformisation problem [FJLW16]. An (implementation) function $f : \mathsf{In} \to \mathsf{Out}$ is said to *uniformise* a (specification) relation $R \subseteq \mathsf{In} \times \mathsf{Out}$ whenever:

(1) $\mathrm{dom}(f) = \mathrm{dom}(R)$ and
(2) for all $i \in \mathrm{dom}(f), (i, f(i)) \in R$

Note that constraint 1 is the main difference with the notion of realisability.

In the context of reactive synthesis, where $f = f_I$ is defined from an implementation I and R is given as a language of relational words, it can be rephrased as

(1) $\mathsf{inp}(L(I)) = \mathsf{inp}(R)$ and

(2) for all $w_{\mathbb{i}} \in \mathsf{inp}(L(I)), \langle w_{\mathbb{i}}, f_I(w_{\mathbb{i}})\rangle \in R$

Note that such definition coincides with the one of realisability of Section 2 when the class of implementations has total domain, because then it is equivalent to asking $L(I) \subseteq R$. In the following, we denote by $\textsc{Unif}(\mathcal{S}, \mathcal{I})$ the uniformisation problem from specifications in $\mathcal{S}$ to implementations in $\mathcal{I}$. Unfortunately, this setting is actually much harder, as shown by the next two theorems:

Theorem 5.1. *Given S a specification represented by a* DRA*, checking whether $\mathsf{inp}(S) = \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$ is undecidable.*

Proof. We reduce from the universality problem of NRA, which is undecidable [NSV04]. Let $A = (\Sigma, \mathcal{D}, Q, q_0, \delta, R, c)$ be an NRA. We encode $L(A)$ as the domain of some DRA specification: the input transitions are the same as the transitions of the original automaton, but when there is some nondeterminism, its resolution is postponed to the corresponding output transition, whose finite label corresponds to the chosen transition. In the vocabulary of games, the input player chooses the finite input label and the equality relation of the input data to the registers of A, and the output player resolves the nondeterminism. Thus, we construct a DRA D accepting $R(D) = \{((\sigma_1, d_1)(\sigma_2, d_2)\dots, (t_1, d_1)(t_2, d_2)\dots) \mid t_1 t_2 \dots$ is a run of A over $(\sigma_1, d_1)(\sigma_2, d_2)\dots\}$.

Thus, define $D = (\Sigma \uplus \delta, \mathcal{D}, Q \uplus Q \times (\Sigma \times \mathsf{Tst}_R), q_0, \delta', R \uplus \{r_0\}, c')$, where δ' is defined as follows: for all $q \in Q$, $\sigma \in \Sigma$ and $\phi \in \mathsf{Tst}_R$, we define the input transition $q \xrightarrow[D]{\sigma,\phi,\{r_0\}} (q, (\sigma, \phi))$. Then, for all $t = q \xrightarrow[A]{\sigma,\phi,\mathsf{asgn}} q' \in \delta$, we define the output transition $(q, (\sigma, \phi)) \xrightarrow[D]{t,\phi\wedge r_0^{=},\mathsf{asgn}} q'$. Then, let $c' : q \mapsto c(q)$ and $(q, \bullet) \mapsto c(q)$. Such automaton is indeed deterministic, and it recognises the relation $R(D) = \{((\sigma_1, d_1)(\sigma_2, d_2)\dots, (t_1, d_1)(t_2, d_2)\dots) \mid t_1 t_2 \dots$ is a run of A over $(\sigma_1, d_1)(\sigma_2, d_2)\dots\}$. Then, $\mathsf{inp}(R(D))$ is universal iff $L(A)$ is universal. □

Such result extends to NRA and URA, whose DRA are a special case. Note that the unbounded realisability problem for DRA is not reducible to deciding whether the domain is total: if the specification S is not realisable, it is not possible to determine whether it is because the domain of S is not total or because S is not realisable by a sequential machine (e.g. S asks to output right away a data that will only be input in the future).

Then, while the uniformisation setting obviously preserves the undecidability results from the synthesis setting, the above result allows to show that the somehow more general uniformisation problem is undecidable. For instance, we can prove:

Theorem 5.2. *For all $k \geq 1$, $\textsc{Unif}(\mathsf{URA}, \mathsf{RT}[k])$ is undecidable.*

Proof. Consider some unrealisable URA specification S_u and the following specification S mapping $w_1 \# w_2$ to $w_1 \# w_2'$ such that $(w_2, w_2') \in S_u$, defined only when w_1 is a finite data word accepted by some URA A. Clearly, S is URA-definable and realisable iff its domain is empty, i.e. $L(A) = \varnothing$. However, emptiness of URA is an undecidable problem. □

If the domain of the specification is DRA-recognisable, it is possible to reduce the uniformisation problem to realisability, by allowing any behaviour on the complement of the domain (which is then DRA-recognisable). However, such property is undecidable as a direct corollary of Theorem 5.1.

Conclusion

In this paper, we have given a picture of the decidability landscape of the synthesis of register transducers from register automata specifications. We studied the parity acceptance condition because of its generality, but our results allow to reduce the synthesis problem for register automata specifications to the one for finite automata while preserving the acceptance condition. We have also introduced and studied test-free NRA, which do not have the ability to test their input, but still have the power of duplicating, removing or copying the input data to form the output. We have shown that they allow to recover decidability in the presence of non-determinism, in the bounded synthesis case. We leave open the unbounded case, which we conjecture to be decidable. As future work, we want to study synthesis problems for register automata which are able to test additional properties over the data. In particular, allowing to compare data for an order over $\mathcal{D}$ [BLP10b, FHL16] looks promising. Note that most other natural predicates immediately yield undecidability, e.g. adding $+1$. Another direction is to study specifications given by logical formulae, for decidable data words logics such as two-variable fragments of FO [BMS$^+$06, SZ12, DFL18]. Such problem is however much more challenging, as there do not exist good correspondence between logic and automata in the realm of data words, except in very restricted settings [BLP10a].

Acknowledgments

The authors would like to thank Ayrat Khalimov for his remarks and suggestions, which helped improve the quality of the paper. They also thank the anonymous reviewers, who took the time to read the paper in detail and subsequently suggested important clarifications as well as simplifications in the proofs.

References

[AG11] Krzysztof R. Apt and Erich Grädel. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[BCJ18] Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. *Graph Games and Reactive Synthesis*, pages 921–962. Springer International Publishing, Cham, 2018.

[BL69] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[BLP10a] Michael Benedikt, Clemens Ley, and Gabriele Puppis. Automata vs. logics on data words. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2010.

[BLP10b] Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. In Alberto H. F. Laender and Laks V. S. Lakshmanan, editors, *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, Buenos Aires, Argentina, May 17-20, 2010*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[BMS$^+$06] Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-Variable Logic on Words with Data. In *Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 7–16. ACM, 2006.

[CJK$^+$17] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 252–263. ACM, 2017.

[CKS81] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.

[Cou94] Bruno Courcelle. Monadic second-order definable graph transductions: A survey. *Theor. Comput. Sci.*, 126(1):53–75, April 1994.

[DFL18] Luc Dartois, Emmanuel Filiot, and Nathan Lhote. Logics for Word Transductions with Synthesis. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 295–304. ACM, 2018.

[DH16] Antoine Durand-Gasselin and Peter Habermehl. Regular Transformations of Data Words Through Origin Information. In *Proceedings of the 19th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2016)*, volume 9634 of *Lecture Notes in Computer Science*, pages 285–300. Springer, 2016.

[DL09] Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009.

[EFR19] Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of data word transducers. In *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 24:1–24:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[ESK14] Rüdiger Ehlers, Sanjit A. Seshia, and Hadas Kress-Gazit. Synthesis with identifiers. In *Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2014)*, volume 8318 of *Lecture Notes in Computer Science*, pages 415–433. Springer, 2014.

[FHL16] Diego Figueira, Piotr Hofman, and Slawomir Lasota. Relating timed and register automata. *Math. Struct. Comput. Sci.*, 26(6):993–1021, 2016.

[FJLW16] Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *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 fuer Informatik, 2016.

[FP18] Diego Figueira and M. Praveen. Playing with Repetitions in Data Words Using Energy Games. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 404–413. ACM, 2018.

[KF94] Michael Kaminski and Nissim Francez. Finite-memory Automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

[KMB18] Ayrat Khalimov, Benedikt Maderbacher, and Roderick Bloem. Bounded synthesis of register transducers. In *Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis (ATVA 2018)*, volume 11138 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2018.

[KZ10] Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *Int. J. Found. Comput. Sci.*, 21(5):741–760, 2010.

[LTV15] Leonid Libkin, Tony Tan, and Domagoj Vrgoc. Regular expressions for data words. *J. Comput. Syst. Sci*, 81(7):1278–1297, 2015.

[NSV04] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite State Machines for Strings over Infinite Alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.

[Pit07] Nir Piterman. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. *Logical Methods in Computer Science*, 3(3), 2007.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM Symposium on Principles of Programming Languages, POPL*. ACM, 1989.

[Seg06] Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Proceedings of the 15th Annual Conference of the EACSL on Computer Science Logic (CSL 2006)*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006.

[SZ12] Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. *Logical Methods in Computer Science*, 8(1), 2012.

This work is licensed under the Creative Commons Attribution License. To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, 171 Second St, Suite 300, San Francisco, CA 94105, USA, or Eisenacher Strasse 2, 10777 Berlin, Germany