HAL
open science

Synthesis of Data Word Transducers

Léo Exibard, Emmanuel Filiot, Pierre-Alain Reynier

▶ **To cite this version:**

Léo Exibard, Emmanuel Filiot, Pierre-Alain Reynier. Synthesis of Data Word Transducers. CONCUR 2019, Aug 2019, Amsterdam, Netherlands. 10.4230/LIPIcs.CONCUR.2019.24 . hal-02439660

HAL Id: hal-02439660

https://hal.science/hal-02439660

Submitted on 14 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Synthesis of Data Word Transducers

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

Emmanuel Filiot
Université libre de Bruxelles, Brussels, Belgium

Pierre-Alain Reynier
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

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 non-deterministic specifications, while decidability is recovered in the deterministic case. In the bounded setting, undecidability still holds for non-deterministic 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.

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

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

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.24

Related Version An extended version can be found at `https://arxiv.org/abs/1905.03538`.

Funding *Léo Exibard*: Funded by a FRIA fellowship from the F.R.S.-FNRS.
Emmanuel Filiot: 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.
Pierre-Alain Reynier: Partly funded by the DeLTA project (ANR-16-CE40-0007).

Acknowledgements We would like to thank Ayrat Khalimov for his remarks and suggestions, which helped improve the quality of the paper.

1 Introduction

Reactive synthesis is an active research domain whose goal is to design algorithmic methods able to 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 *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, 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 $(\mathsf{In}.\mathsf{Out})^\omega$. Classically, the sets In and Out are *assumed to be finite* and reactive

© Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier;
licensed under Creative Commons License CC-BY

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 $S \subseteq (\mathsf{In}.\mathsf{Out})^\omega$ 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 (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 ressource) 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, ExpTime-c when represented by parity automata [9] and 2ExpTime-c for LTL specifications [16]. Such positive results have triggered a recent and very active research interest in efficient symbolic methods and tools for reactive synthesis (see e.g. [1]). Extensions of this classical setting have been proposed to capture more realistic scenarios [1]. 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, and 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 $x_1 x_2 \ldots$ of labelled data, i.e., pairs (σ, d) with σ a label from a finite alphabet, and $d \in \mathbb{N}$. 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 [10, 18]. 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 together with the content of one of its registers. Its executions are then data words alternating between input and output labelled data, so register automata can be used to represent specifications, as languages of such data words.

Contributions. We consider two classical semantics for register automata, non-deterministic and universal, both with a parity acceptance condition, which give two classes of register automata respectively denoted NRA and URA. Since NRA are not closed under complement (already over finite data words), NRA and URA define incomparable classes of specifications. The request-grant specification given previously with an infinite number of clients is expressible by an URA [12]: 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; but no NRA can define it. On the contrary, 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 (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 [12]. On the positive side, we show that decidability is recovered for deterministic (parity) register automata (DRA). 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 DRA: any specification expressed as a DRA 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 [12], which establishes that bounded synthesis is decidable in 2ExpTime for URA. We show that unfortunately, bounded synthesis is still undecidable for NRA specifications. 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 [12], 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.

Table 1 Decidability status of the problems studied.

	DRA	NRA	URA	$\mathsf{NRA_{tf}}$
Bounded Synthesis	ExpTime (Thm. 13)	Undecidable ($k \geq 1$) (Thm. 3)	2ExpTime ([12] and Thm. 12)	2ExpTime (Thm. 16)
General Case	ExpTime (Thm. 6)	Undecidable (Thm. 2)	Undecidable (Thm. 4)	Open

Related Work. As already mentioned, bounded synthesis of register transducers is considered in [12] where it is shown to be decidable for URA. We reprove this result in a shorter way. Our proof bears some similarities with that of [12], 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, non-deterministic version of the expressive transducer model of [5], which however does not consider the synthesis problem.

The synthesis problem over infinite alphabets is also considered in [6], 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 [8]. Such games are shown to be undecidable and a decidable restriction is proposed, which however does not seem to match our context.

2 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* $\langle u, v\rangle = u[1]v[1]u[2]v[2]\dots$

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.

Register Automata. Register automata are automata recognising data word languages. They were first introduced in [10] as finite-memory automata. Here, we define them in a spirit close to [18], but over infinite words [1], with a parity acceptance condition.

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 $\mathsf{Tst}_R = 2^R$ and $\mathsf{Asgn}_R = 2^R$. [2]
- $c : Q \to \{1, \dots, d\}$, 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 $(q, \sigma, \mathsf{tst}, \mathsf{asgn}, q')$ is also written $q \xrightarrow[\mathcal{A}]{\sigma,\mathsf{tst},\mathsf{asgn}} q'$. We may omit $\mathcal{A}$ in the latter notation. Intuitively such transition means that on input (σ, d) in state q the automaton: it first checks that tst is *exactly* the set of registers containing d: for all $r \in \mathsf{tst}$, d is the current content of register r and for all $r \notin \mathsf{tst}$, d is not in register r, then it assigns d to all the registers in asgn (asgn might be empty), and finally transitions to state q'.

$\mathcal{A}$ is said to be *deterministic* (resp. *complete*) when, given any state, any label and any possible test, at most (resp. at least) one transition can be taken: $\forall q \in Q, \forall \sigma \in \Sigma, \forall \mathsf{tst} \in \mathsf{Tst}_R, \exists^{\leq 1} \mathsf{asgn} \in \mathsf{Asgn}, \exists^{\leq 1} q' \in Q$ such that $q \xrightarrow{\sigma,\mathsf{tst},\mathsf{asgn}} q'$ (resp. $\forall q, \forall \sigma, \forall \mathsf{tst}, \exists^{\geq 1} \mathsf{asgn}, \exists^{\geq 1} q'$ s.t. $q \xrightarrow{\sigma,\mathsf{tst},\mathsf{asgn}} q'$). Since tests are mutually exclusive, this syntactically ensures that for any state and in any register configuration, the transition to take is determined by the input l-data (respectively that a transition can always be taken, regardless the input l-data). The class of deterministic register automata will be denoted DRA.

[1] In the terminology of [13], it corresponds to multiple-assignment register automata with registers initially filled, i.e. the class MF, with the additional requirement that all are filled with the same data.

[2] Those sets are identical but have distinct semantics.

Configurations and Runs. A configuration is a pair $(q, \tau) \in Q \times (R \to \mathcal{D})$. Given $\mathsf{tst} \in \mathsf{Tst}_R$ and $d \in \mathcal{D}$, we say that τ, d satisfies tst, denoted $\tau, d \models \mathsf{tst}$ if $\tau^{-1}(d) = \mathsf{tst}$. Given a transition $t = p \xrightarrow{\sigma,\mathsf{tst},\mathsf{asgn}} p'$, we say that (q, τ) enables t on reading (σ', d) if $q = p$, $\sigma' = \sigma$ and $\tau, d \models \mathsf{tst}$. Let $\mathrm{next}(\tau, \mathsf{asgn}, d)$ be the configuration 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 $\mathrm{next}(\gamma, \mathsf{asgn}, d) = (p', \mathrm{next}(\tau, \mathsf{asgn}, d))$. 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 $\gamma_0\gamma_1 \dots = (q_0, \tau_0)(q_1, \tau_1)\dots$ such that γ_0 is initial and for all $i \geq 0$, $\gamma_{i+1} = \mathrm{next}(\gamma_i, t_i, \sigma_i, d_i)$. To a run ρ, we associate its sequence of states $\mathsf{states}(\rho) = q_0 q_1 \dots$

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: non-deterministic (N) and universal (U). Given an 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 $L_U(A) = \{w \mid \text{all runs } \rho \text{ on } w \text{ are accepting}\}$.

We denote by NRA (resp. URA) the class of register automata interpreted with a non-deterministic (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.

3 Synthesis of Register Transducers

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 $w \in [(\Sigma_{\mathbb{i}} \times \mathcal{D}).(\Sigma_{\mathbb{o}} \times \mathcal{D})]^\omega$. 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 $w = x_{\mathbb{i}}^1 x_{\mathbb{o}}^1 x_{\mathbb{i}}^2 x_{\mathbb{o}}^2 \dots$, we let $\mathsf{inp}(w) = x_{\mathbb{i}}^1 x_{\mathbb{i}}^2 \dots$ and $\mathsf{out}(w) = x_{\mathbb{o}}^1 x_{\mathbb{o}}^2 \dots$ 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 $I : (\Sigma_{\mathbb{i}} \times \mathcal{D})^* \to \Sigma_{\mathbb{o}} \times \mathcal{D}$. We associate to I another function $f_I : \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D}) \to \mathsf{DW}(\Sigma_{\mathbb{o}}, \mathcal{D})$ which, to an input data word $w_{\mathbb{i}} = x_{\mathbb{i}}^1 x_{\mathbb{i}}^2 \dots \in \Sigma_{\mathbb{i}} \times \mathcal{D}$, associates the output data word $f_I(w_{\mathbb{i}}) = x_{\mathbb{o}}^1 x_{\mathbb{o}}^2 \dots$ such that $\forall i \geq 1$, $x_{\mathbb{o}}^i = I(x_{\mathbb{i}}^1 \dots x_{\mathbb{i}}^{i-1})$. 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. 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.

Specification Register Automata. In this paper, we consider specifications defined from register automata alternately reading input and output l-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 $F \subseteq Q_{\mathbb{i}}$, and such that the

transition relation δ alternates between these two sets, i.e. $\delta \subseteq \cup_{\alpha=\mathbb{i},\mathbb{o}}(Q_\alpha \times \Sigma_\alpha \times \mathsf{Tst}_R \times \mathsf{Asgn}_R \times Q_{\overline{\alpha}})$, 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. non-deterministic, universal) parity register automata.

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 (total) transition function (as before, $\mathsf{Tst}_R = \mathsf{Asgn}_R = 2^R$). 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 $w = (\sigma^1_\mathbb{i}, d^1_\mathbb{i}) \dots (\sigma^n_\mathbb{i}, d^n_\mathbb{i})$ 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 $\delta(q_{i-1}, \sigma^i_\mathbb{i}, \mathsf{tst}) = (\mathsf{asgn}, \sigma_\mathbb{o}, r, q_i)$ such that $\tau_{i-1}, d^i_\mathbb{i} \models \mathsf{tst}$ and $\tau_i = \mathrm{next}(\tau_{i-1}, d^i_\mathbb{i}, \mathsf{asgn})$. We let $(\sigma^i_\mathbb{o}, d^i_\mathbb{o}) = (\sigma_\mathbb{o}, \tau_i(r))$ and $I_T(w) = (\sigma^n_\mathbb{o}, d^n_\mathbb{o})$. 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 2), 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.

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

▶ **Theorem 1** ([9]). *Given a (data-free) specification S defined by some (register-free) universal or non-deterministic parity automaton, the realisability problem of S by (register-free) transducers is* ExpTime-c.

4 Unbounded Synthesis

In this section, we consider the unbounded synthesis problem $\textsc{Real}(\mathsf{RA}, \mathsf{RT})$. Thus, we do not fix a priori the number of registers of the implementation. Let us first consider the case of NRA and URA, which are, in our setting, the most natural devices to express data word specifications. By reducing the universality of NRA over finite words (which is undecidable [14]) to our synthesis problems, we show:

▶ **Theorem 2.** $\textsc{Real}(\mathsf{NRA}, \mathsf{RT})$ *is undecidable.*

▶ **Theorem 3.** *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 [12].

▶ **Theorem 4.** $\textsc{Real}(\mathsf{URA}, \mathsf{RT})$ *is undecidable.*

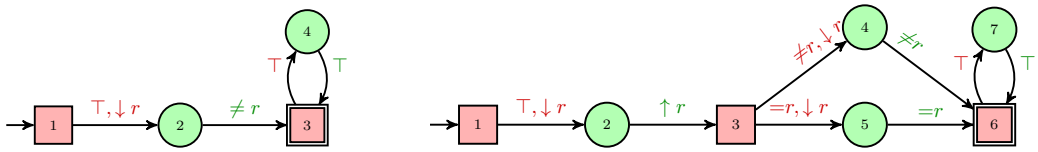
Proof. We present a reduction from the emptiness problem of URA over finite words (which is undecidable by a direct reduction from the universality problem of NRA, undecidable by [14]) to our synthesis problem.

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 $|\mathsf{dt}(u)| \leq k\}$ is realisable by a k-register transducer: on reading u, store each distinct data in one register, and after the # outputs 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 $\mathsf{dt}(u)$ in its registers, but the size of $\mathsf{dt}(u)$ is not bounded (u can have pairwise distinct data and be of arbitary 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 [10], 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 k its number of distinct data. Then S is realisable by a k-register transducer realising S_1 when the number of data in u is lower than or equal to k, and, when it is greater than k, by outputting $u\#\widehat{w}\#(a, \mathsf{d}_0)^\omega$ where $\widehat{w}$ is a membership-preserving renaming of w using k distinct data of u (this can always be done thanks to the so-called "indistinguishability property" stated in [10]). 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 $L(A) = \varnothing$. ◀

However, we show that restricting to DRA allows to recover the decidability, modulo one additional assumption, namely that no output transition of the specification is such that $\mathsf{tst} = \varnothing$ (the output data is different from all register contents). We denote by $\mathsf{DRA}_\varnothing$ this class of DRA. 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. 1a (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 (with $\mathsf{tst} = \varnothing$ depicted here $\neq r$), 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 $\mathsf{tst} = \varnothing$ does not imply unrealisability, as shown by the DRA of Fig. 1b: 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 1 Pathological DRA specifications.

To check for realisability of $\mathsf{DRA}_\varnothing$-specifications with r-registers, we show that it suffices to target transducers with r registers only.

▶ **Proposition 5.** *Let S be a specification defined by a $\mathsf{DRA}_\varnothing$ with r registers. If S is realisable by a register transducer, then it is realisable by a transducer with r registers.*

Proof. Let $S = (\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D}, Q^S, q_0^S, \delta^S, R^S, c)$ be a $\mathsf{DRA}_\varnothing$ specification realisable by a register transducer $I = (\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D}, Q^I, q_0^I, \delta^I, R^I)$. Without loss of generality, we assume that S does not conduct assignments on the output and that S is complete. Now, from S, we extract a transducer I' realising S with R^S registers, using I as a guide to make choices for the output. To this end, we simulate I synchronously with S. However, we cannot properly simulate I, since we only have R^S registers, which are used to simulate S. Instead, we keep *constraints* in memory.

Constraints. A constraint represents the equality relations between the registers in R^S and those in R^I (note that such idea is pervasive in the study of registers automata, e.g. to recognise the projection over finite labels). Thus, a constraint is a subset $C \subseteq R^S \times R^I$, which is intended to be a set of equalities between the content of registers in R^S and in R^I. Then, knowing tests $\mathsf{tst}^S, \mathsf{tst}^I$ and assignments asgn^S, asgn^I performed by S and I respectively allows to update the constraints: we define

$$\begin{aligned}\mathrm{next}(C, \mathsf{tst}^S, \mathsf{asgn}^S, \mathsf{tst}^I, \mathsf{asgn}^I) = C \backslash ((\mathsf{asgn}^S \times R^I) \cup (R^S \times \mathsf{asgn}^I)) \\ \cup ((\mathsf{tst}^S \cup \mathsf{asgn}^S) \times (\mathsf{tst}^I \cup \mathsf{asgn}^I))\end{aligned}$$

For instance, assume $R^I = \{r_1, r_2\}$ and $R^S = \{s_1, s_2\}$, and at some point in a run, we have[3] $C = \{(s_2, r_1), (s_2, r_2)\}$, i.e. $s_2 = r_1 = r_2$ and $s_1 \neq r_1$ (inequalities are implicit, since C is an exhaustive list of equalities). Now, S reads some data d which satisfies the tests $\mathsf{tst}^S = \{s_1\}$ in S and $\mathsf{tst}^I = \varnothing$ in I (such tests are consistent because $s_1 \neq r_1, r_2$), and conducts assignments $\mathsf{asgn}^S = \varnothing$ and $\mathsf{asgn}^I = \{r_2\}$. Then, on the one hand, $s_1 = r_2$ (both contain d), and on the other hand $s_2 = r_1$ (since the content of those registers did not change). Moreover, $r_1 \neq r_2$ since r_2 has been reassigned and $s_1 \neq r_1$ still holds. This is represented by the set of constraints $C' = \{(s_1, r_2), (s_2, r_1)\}$, and indeed, $\mathrm{next}(C, \{s_1\}, \varnothing, \varnothing, \{r_2\}) = C'$.

Abstracting the behaviour of I modifies its language (it somehow simplifies it), but we will see that what we build is still an implementation.

Definition of I'. We build $I' = (\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D}, Q, q_0, \delta, R^S)$, where $Q = Q^S \times Q^I \times 2^{R^S \times R^I}$ and $q_0 = (q_0^S, q_0^I, R^S \times R^I)$; we now define δ. For each state $(q_{\mathbb{i}}^S, q^I, C) \in Q$, for each input test $(\sigma_{\mathbb{i}}, \mathsf{tst}_{\mathbb{i}}^S) \in \Sigma_{\mathbb{i}} \times \mathsf{Tst}_{R^S}$, we construct a transition $t = (q_{\mathbb{i}}^S, q^I, C) \xrightarrow[I']{\sigma_{\mathbb{i}}, \mathsf{tst}_{\mathbb{i}}^S, \mathsf{asgn}^S, \sigma_{\mathbb{o}}, s_{\mathbb{o}}} (q'^S_{\mathbb{i}}, q'^I, C')$ whenever there exist the following transitions of S and I:

$$t_{\mathbb{i}}^S = q_{\mathbb{i}}^S \xrightarrow[S]{\sigma_{\mathbb{i}}, \mathsf{tst}_{\mathbb{i}}^S, \mathsf{asgn}^S} q_{\mathbb{o}}^S \qquad t_{\mathbb{o}}^S = q_{\mathbb{o}}^S \xrightarrow[S]{\sigma_{\mathbb{o}}, \mathsf{tst}_{\mathbb{o}}^S} q'^S_{\mathbb{i}} \qquad t^I = q^I \xrightarrow[I]{\sigma_{\mathbb{i}}, \mathsf{tst}^I, \mathsf{asgn}^I, \sigma_{\mathbb{o}}, r_{\mathbb{o}}} q'^I$$

such that, for some fixed arbitrary order on R^S, we have:

(i) $\mathsf{tst}^I = \{r \in R^I \mid \exists s \in \mathsf{tst}_{\mathbb{i}}^S, (s, r) \in C\}$

(ii) $C' = \mathrm{next}(C, \mathsf{tst}_{\mathbb{i}}^S, \mathsf{asgn}^S, \mathsf{tst}^I, \mathsf{asgn}^I)$

(iii) $\mathsf{tst}_{\mathbb{o}}^S = \{s \in R^S \mid (s, r_{\mathbb{o}}) \in C'\}$

(iv) $s_{\mathbb{o}} = \min \mathsf{tst}_{\mathbb{o}}^S$

[3] For readability, we confuse a register with its content.

Item (i) ensures with the help of constraints that in any reachable configuration of I', there exists at least one input data which satisfies both $\mathsf{tst}^S_{\mathbb{i}}$ and tst^I, which allows I' to synchronise S with I. Note that this does not means that I' is the synchronous product of S and I *on any input*: since I' only has the registers of S, it cannot discriminate data as subtly as I, and might thus adopt a different behaviour. For instance, it can be that upon reading some input data word, at some point, I would store some input data d in some register r that S would not, and use it later on in a test $\mathsf{tst}^I = \{r\}$ to take different actions, while neither I' nor S could discriminate between those choices: on reading d, I' simulates S with $\mathsf{tst}^S_{\mathbb{i}} = \varnothing$ and synchronously simulates in I the transition with input test $\mathsf{tst}^I = \varnothing$. Nevertheless, we show the *existence* of some relational data word common to I and I' for each run of I' (which is also a run of S). This is sufficient to conclude that I' realises S, because then each run of I', interpreted as a run of S, is accepting. Then, items (iii) and (iv) ensures the same property as item (i) does, but this time on output positions.

We shall see that for a transition t such that (q^S, q^I, C) is accessible on some finite input data word, $t^S_{\mathbb{i}}$, $t^S_{\mathbb{o}}$ and t^I exist and are unique. So, for a run $\rho = t_1 t_2 \dots$ of I', we define $\rho^S = t^S_{\mathbb{i}1} t^S_{\mathbb{o}1} t^S_{\mathbb{i}2} t^S_{\mathbb{o}2} \dots$ and $\rho^I = t^I_1 t^I_2 \dots$

Proof of Correctness. Let us show that I' is indeed a transducer realising S: we show that for all $u_{\mathbb{i}} \in \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$, there exists a unique sequence of transitions ρ in I', a unique output data word $u_{\mathbb{o}} \in \mathsf{DW}(\Sigma_{\mathbb{o}}, \mathcal{D})$ (we denote $u = \langle u_{\mathbb{i}}, u_{\mathbb{o}} \rangle$) and a $w \in L(I)$ such that:

1. ρ^I is the run of I over w
2. ρ^S is the run of S over w
3. ρ is the run of I' over u
4. ρ^S is the run of S over u

Note that the above properties imply $\mathsf{lab}(u) = \mathsf{lab}(w)$, but it can be that $u \neq w$, which is consistent with the observations we made. Let us show that they entail the result we need: let $u_{\mathbb{i}} \in \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$ be some input data word. By property 3, $f_{I'}(u_{\mathbb{i}})$ exists and is unique, so I' has total domain. Now, by denoting ρ^S the run of S over $u = \langle u_{\mathbb{i}}, f_{I'}(u_{\mathbb{i}}) \rangle$, we know by property 2 that there exists $w \in L(I)$ such that ρ^S is the run of S over w. Then, ρ^S is accepting because I realises S so $w \in L(S)$, hence $u \in L(S)$. Thus, I' realises S.

The proof of properties 1-4 is rather technical, and can be found in [7]. ◀

Such result allows us to reduce unbounded synthesis to bounded synthesis for $\mathsf{DRA}_{\varnothing}$. Bounded synthesis is in ExpTime for DRA (Thm. 13) and is the topic of the next section.

▶ **Theorem 6.** Real($\mathsf{DRA}_{\varnothing}, \mathsf{RT}$) *is decidable in* ExpTime.

5 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 3), by reducing the problems to realisability problems for data-free specifications.

Abstract Actions. We reduce the problem to a finite alphabet problem. Since we synthesise k-register transducers, we take the input and output actions of the transducers as symbols of our finite input and output alphabets. Let $R_k = \{1, \dots, k\}$ and $\mathsf{Tst}_k = \mathsf{Asgn}_k = 2^{R_k}$. 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 $\overline{a} = (\sigma_{\mathbb{i}}^1, \mathsf{test}_1)(\sigma_{\mathbb{o}}^1, \mathsf{asgn}_1, r_1) \cdots \in (A_{\mathbb{i}}^k A_{\mathbb{o}}^k)^\omega$ 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 $i \geq 1$, $\tau_i, d_{\mathbb{i}}^i \models \mathsf{test}_i$, $d_{\mathbb{o}}^i = \tau_i(r_i)$ and $\tau_{i+1} = \mathrm{next}(\tau_i, d_{\mathbb{i}}^i, \mathsf{asgn}_i)$. 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 $W_{S,k}$ can be seen as a specification over the finite input (resp. output) alphabets $A_{\mathbb{i}}^k$ (resp. $A_{\mathbb{o}}^k$).

▶ **Theorem 7** (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 transducer T can be seen as a finite transducer T' over input alphabet $A_{\mathbb{i}}^k$ and output alphabet $A_{\mathbb{o}}^k$. Indeed, since the transition function of T is total, 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 $\overline{a} = a_1e_1a_2e_2 \cdots \in L(T')$. We show that $\mathsf{Comp}(\overline{a}) \subseteq S$. Let $w \in \mathsf{Comp}(\overline{a})$. Then, there exists a run $q_0q_1q_2 \ldots$ 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 $(q_0, \tau_0)(q_1, \tau_1) \ldots$ is an initial sequence of configurations of T over w, so $w \in L(T)$. Finally, $L(T) \subseteq S$, since T realises 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 with k registers over data words. 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. ◀

5.1 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 $\textsc{Real}(\mathsf{URA}, \mathsf{RT}[k])$, by Theorem 7 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 $(w \otimes \overline{a})[i] = (\overline{a}[i], \mathsf{dt}(w[i]))$.

▶ **Lemma 8.** *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 $(\sigma_{\mathbb{i}}, \mathsf{tst}, d)$, it guesses some assignment asgn, performs

the test tst 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 $(\sigma_{\mathbb{D}}, \mathsf{asgn}, r, d)$, 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 maximal even parity 2). 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 9.** *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 the same automaton, denoted now $A_{\overline{S}}$, interpreted as an NRA. Let B be some NRA defining L_k (it exists by Lemma 8). 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 10.** $W_{S,k} = \overline{\mathsf{lab}(L_{\overline{S},k})}$.

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 regularity of $W_{S,k}$.

▶ **Lemma 11.** *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 some deterministic parity automaton with $O(2^{n^2 16^{(r+k)^2}})$ states and $O(n.4^{(r+k)^2})$ 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 10, obtained as product between the NRA $A_{\overline{S}}$ and the automaton obtained in Lemma 8, 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 [10]. The same construction, which is based on extending the state space with register equality types, carries over to ω-words, and one obtains a non-deterministic parity automaton with $O(n.4^{(r+k)^2})$ states and d priorities recognising $\mathsf{lab}(L_{\overline{S},k})$. It can be complemented into a deterministic parity automaton with $O(2^{n^2.16^{(r+k)^2}})$ states and $O(n.4^{(r+k)^2})$ priorities using standard constructions [15]. ◀

We are now able to reprove the following result, known from [12]:

▶ **Theorem 12.** *For all $k \geq 0$,* $\textsc{Real}(\mathsf{URA}, \mathsf{RT}[k])$ *is in* 2ExpTime.

Proof. By Lemma 11, we construct a deterministic parity automaton $P_{S,k}$ for $W_{S,k}$. Then, according to Theorem 7, it suffices to check whether it is realisable by a (register-free) transducer. This is decidable by Theorem 1. The way to decide it is to see $P_{S,k}$ as a two-player parity game and check whether the protagonist has a solution. Parity games can be solved in time $O(m^{\log d})$ [3] 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 $O(2^{n^3 16^{(r+k)^2}})$. ◀

In [11], it is shown that the complexity of the problem is actually only singly exponential in k, and such analysis extends to our construction. Similarly, when the specification automaton is deterministic, we can show that:

▶ **Theorem 13.** $\textsc{Real}(\mathsf{DRA}, \mathsf{RT}[k])$ *is in* ExpTime.

5.2 The case of test-free NRA specifications

Unfortunately, by Theorem 3, 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, Lemma 10 does not allow to conclude because $L_{\overline{S},k}$ is definable by a URA but the string projection of a URA is not ω-regular in general. E.g., consider $L = \{(r, d_1) \dots (r, d_n)(g, d'_1) \dots (g, d'_m) \mid \forall i \neq j, d_i \neq d_j \wedge \forall 1 \leq i \leq n, \exists j, d'_j = d_i\}$, 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 (it is over finite words for simplicity but can be extended to ω-words). L is recognised by the URA which, on reading (r, d_i), universally triggers a run checking three properties: firstly, once a label g is read, only g's are read, secondly, (r, d_i) does not appear again, and thirdly, (g, d_i) appears at least once. Now, it is readily seen that $\mathsf{lab}(L) = \{r^n g^m \mid m \geq n\}$, which is not regular. Here, we defined L over finite words for simplicity but it is easily extended to an ω-language which is not ω-regular.

Thus, in this section, we consider a restriction on NRA which do not perform tests on input data. A *test-free register automaton* is a tuple $A = (\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D}, Q, q_0, \delta, R, c)$ such that $\delta \subseteq Q \times \Sigma_{\mathbb{i}} \times \mathsf{Asgn}_R \times \Sigma_{\mathbb{o}} \times R \times Q$. Such a register automaton reads two labelled data at once. In a configuration (q, τ), when reading $(\sigma_{\mathbb{i}}, d_{\mathbb{i}})(\sigma_{\mathbb{o}}, d_{\mathbb{o}})$, it can fire any transition of the form $(q, \sigma_{\mathbb{i}}, \mathsf{asgn}, r, \sigma_{\mathbb{o}}, q') \in \delta$ such that $\tau(r) = d_{\mathbb{o}}$ and move to configuration (q', τ') where $\tau' = \mathrm{next}(\tau, \mathsf{asgn}, d_{\mathbb{i}})$. It is easily seen that a test-free register automaton can be converted into a proper register automaton, justifying its name. Such automata will be interpreted by a non-deterministic parity acceptance condition; we denote the class $\mathsf{NRA}_{\mathsf{tf}}$[4].

It is not clear whether $W_{S,k}$ is regular for such 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), and based on the behaviour of S over input with pairwise distinct data. The intuition behind restricting to such case is that $\mathsf{NRA}_{\mathsf{tf}}$ cannot conduct test on input data, so they behave the same on an input word whose data are all distinct, and such choice ensures that two equal input data will not ease the task of the implementation. 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; formally $\delta : Q \times \Sigma_{\mathbb{i}} \to \mathsf{Asgn}_R \times \Sigma_{\mathbb{o}} \times R \times Q$. In the following, we let AllDiff denote the set of relational data words whose input data are pairwise distinct: $\textsc{AllDiff} = \{w = (\sigma^1_{\mathbb{i}}, d^1_{\mathbb{i}})(\sigma^1_{\mathbb{o}}, d^1_{\mathbb{o}}) \dots \in \mathsf{RW} \mid \forall 0 \leq i < i', d^i_{\mathbb{i}} \neq d^{i'}_{\mathbb{i}}\}$; by convention $d^0_{\mathbb{i}} = \mathsf{d}_0$.

▶ **Proposition 14.** *Let S be a $\mathsf{NRA}_{\mathsf{tf}}$ specification. The following are equivalent:*

(i) *S is realisable*

(ii) *$W^{\mathsf{tf}}_{S,k} = \{\overline{a} \in (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega \mid \mathsf{Comp}(\overline{a}) \cap S \cap \textsc{AllDiff} \neq \varnothing\}$, where $A^{\varnothing}_{\mathbb{i}} = \Sigma_{\mathbb{i}} \times \{\varnothing\}$, has domain $(A^{\varnothing}_{\mathbb{i}})^\omega$ and is realisable (by a register-free transducer)*

(iii) *S is realisable by a test-free transducer*

Proof. (i) $\Rightarrow$ (ii): If S is realisable, then by Theorem 7 $W_{S,k}$ has total domain and is realisable by some transducer I. Now, since transducers are closed under regular domain restriction, $W^{\varnothing}_{S,k} = W_{S,k} \cap (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$ has domain $(A^{\varnothing}_{\mathbb{i}})^\omega$ and is realisable by $I \cap (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$. Moreover, $W^{\varnothing}_{S,k} \subseteq W^{\mathsf{tf}}_{S,k}$. Indeed, if $\mathsf{Comp}(\overline{a}) \subseteq S$, then, since S has total domain and $\overline{a} \in (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$, $\mathsf{Comp}(\overline{a}) \cap S \cap \textsc{AllDiff} \neq \varnothing$. Thus, $W^{\mathsf{tf}}_{S,k}$ also has domain $(A^{\varnothing}_{\mathbb{i}})^\omega$ and is realisable by any transducer realising $W^{\varnothing}_{S,k}$.

[4] The bounded synthesis of URA is already decidable, so we do not consider their test-free restriction.

(iii) $\Rightarrow$ (i): is trivial.

(ii) $\Rightarrow$ (iii): Intuitively, $\mathsf{NRA}_{\mathsf{tf}}$ can only rearrange input data (duplicate, erase, copy) regardless of the actual data values (as there are no tests), so its behaviour on AllDiff determines its behaviour on the entire domain. This can be formalised precisely through a notion of data origins given a run (this notion is also made explicit in [5]).

To a run $\rho = q_0 \xrightarrow{\sigma_{\mathbb{i}}^1, \mathsf{asgn}^1, r^1, \sigma_{\mathbb{o}}^1} q_1 \xrightarrow{\sigma_{\mathbb{i}}^2, \mathsf{asgn}^2, r^2, \sigma_{\mathbb{o}}^2} q_2 \dots$ corresponds the origin function $o_\rho : j \mapsto \max\{i \leq j \mid r_j \in \mathsf{asgn}_i\}$, with the convention $\max \varnothing = 0$.

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 $\mathsf{dt}(\mathsf{inp}(w)[0]) = \mathsf{d}_0$.

The following lemma 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:

▶ **Lemma 15.** *Let $w \in RW$ and ρ a sequence of transitions of some $\mathsf{NRA}_{\mathsf{tf}}$. Then,*

(i) *If ρ is a run over w, then $w \models o_\rho$.*

(ii) *If ρ is a run over w and $w \in$ AllDiff, then for all $o : \mathbb{N}\backslash\{0\} \to \mathbb{N}$, $w \models o \Leftrightarrow o = o_\rho$.*

(iii) *If w and ρ have the same finite labels and if $w \models o_\rho$, then ρ is a run over w.*

Proof. (i) and (iii) follow from the semantics of $\mathsf{NRA}_{\mathsf{tf}}$, which do not conduct any test on the input data. The $\Leftarrow$ direction of (ii) is exactly (i). 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 (i) 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, $o = o_\rho$. ◀

Now, assume $W_{S,k}^{\mathsf{tf}}$ 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 transitions $(\sigma_{\mathbb{i}}, \varnothing)$ are now simply labelled $\sigma_{\mathbb{i}}$. 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 $A_{\mathbb{i}}^{\varnothing}$ with same finite labels as w. Let $\overline{a} = I(\overline{a}_{\mathbb{i}})$, and let $w' \in \mathsf{Comp}(\overline{a}) \cap S \cap$ AllDiff (such w' exists because $W_{S,k}^{\mathsf{tf}}$ has domain $(A_{\mathbb{i}}^{\varnothing})^\omega$ and I realises $W_{S,k}^{\mathsf{tf}}$). Then, since $\mathsf{lab}(w) = \mathsf{lab}(w')$, they admit the same run ρ^I in I, so $w, w' \models o_{\rho^I}$. Then, $w' \in S$, so it admits an accepting run ρ^S in S, which implies $w' \models o_{\rho^S}$. Moreover, $w' \in$ AllDiff so, by Lemma 15 ii, we get $o_{\rho^I} = o_{\rho^S}$. Therefore, $w \models o_{\rho^S}$, so, by iii, 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. *End of proof of Prop. 14* ◀

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

Then, $L_{S,k}^{\mathsf{tf}} = \{w \otimes \overline{a} \in \mathsf{RW} \otimes (A_{\mathbb{i}}^{\varnothing} A_{\mathbb{o}}^k)^\omega \mid w \in \mathsf{Comp}(\overline{a}) \cap S^{\varnothing}\}$ is NRA-definable. Indeed, S is $\mathsf{NRA}_{\mathsf{tf}}$-definable, so $S^{\varnothing}$ is NRA-definable, and by Lemma 8, $L_k = \{w \otimes \overline{a} \mid w \in \mathsf{Comp}(\overline{a})\}$ is NRA-definable, so their product recognises $L_{S,k}^{\mathsf{tf}}$. Finally, $W_{S,k}^{\mathsf{tf}} = \mathsf{lab}(L_{S,k}^{\mathsf{tf}})$, and the projection of a NRA over some finite alphabet is regular [10].

Overall, by Theorems 1 and 7, we get (the complexity analysis is the same as for URA):

▶ **Theorem 16.** *For all $k \geq 0$, $\text{Real}(\mathsf{NRA}_{\mathsf{tf}}, \mathsf{RT}[k])$ is decidable and in 2ExpTime.*

6 Synthesis and Uniformisation

In this section, we discuss the relation between realisability and uniformisation of relations: in this paper, if S is realisable by a register transducer, then, in particular, it has universal 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 $\mathsf{inp}(T) = \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$. 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 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 sufficient here. Thus, it is relevant to generalise the realisability problem to the case where the domain of the specification is not universal. 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. A natural way is to consider synthesis as an instance of the uniformisation problem. An (implementation) function $f : I \to O$ is said to *uniformise* a (specification) relation $R \subseteq I \times O$ whenever $\mathrm{dom}(f) = \mathrm{dom}(R)$ and for all $i \in \mathrm{dom}(f), (i, f(i)) \in R$. 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 $\mathsf{inp}(L(I)) = \mathsf{inp}(R)$ and 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 when the class of implementations has universal domain. 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 17.** *Given S a specification represented by a* DRA*, checking whether $\mathsf{inp}(S) = \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$ is undecidable.*

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

If the domain is DRA-recognisable, it is possible to reduce the uniformisation problem to realisability, by allowing any behaviour on the complement on the domain (which is DRA-recognisable). However, such property is undecidable as a direct corollary of Theorem 17.

7 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 presence of non-determinism, in the bounded case. We leave open the unbounded case, which we conjecture to be decidable. As future work, we want to study synthesis problems for specifications given by logical formulae, for decidable data words logics such as two-variable fragments of FO [2, 17, 4].

References

1. Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. *Graph Games and Reactive Synthesis*, pages 921–962. Springer International Publishing, Cham, 2018.
2. 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. doi:10.1109/LICS.2006.51.
3. 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. doi:10.1145/3055399.3055409.
4. 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. doi:10.1145/3209108.3209181.
5. 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. doi:10.1007/978-3-662-49630-5_17.
6. 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. doi:10.1007/978-3-642-54013-4_23.
7. Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of Data Word Transducers. *CoRR*, abs/1905.03538, 2019. arXiv:1905.03538.
8. 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. doi:10.1145/3209108.3209154.
9. J.R. Büchi and L.H. Landweber. Solving Sequential Conditions by Finite-State Strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
10. Michael Kaminski and Nissim Francez. Finite-memory Automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
11. Ayrat Khalimov and Orna Kupferman. Register Bounded Synthesis. In *Proceedings of the 30th International Conference on Concurrency Theory (CONCUR 2019)*, 2019. To appear.
12. 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.
13. Andrzej S. Murawski, Steven J. Ramsay, and Nikos Tzevelekos. Bisimilarity in Fresh-Register Automata. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015)*, pages 156–167. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.24.
14. Frank Neven, Thomas Schwentick, and Victor Vianu. Finite State Machines for Strings over Infinite Alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004. doi:10.1145/1013560.1013562.
15. Nir Piterman. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. *Logical Methods in Computer Science*, 3(3), 2007.
16. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM Symposium on Principles of Programming Languages, POPL*. ACM, 1989.
17. Thomas Schwentick and Thomas Zeume. Two-Variable Logic with Two Order Relations. *Logical Methods in Computer Science*, 8(1), 2012. doi:10.2168/LMCS-8(1:15)2012.
18. 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. doi:10.1007/11874683_3.