Register-Bounded Synthesis from Constraint LTL

Nino Dauvier
Aix Marseille Univ, CNRS, LIS, Marseille, France

Emmanuel Filiot
Université libre de Bruxelles, Belgium

Pierre-Alain Reynier
Aix Marseille Univ, CNRS, LIS, Marseille, France

Abstract

We consider synthesis problems from logical specifications over infinite data domains, expressed in the logic *constraint LTL* (CLTL), which extends LTL with predicates over an infinite set of data values. We consider register-bounded synthesis, where the goal is to automatically generate, if it exists, a transducer with r registers that realizes a given CLTL formula, where r is also given as input. We prove that CLTL register-bounded synthesis is 2ExpTime-c for various data domains such as any infinite set with equality, $(\mathbb{Q}, <)$, and $(\mathbb{N}, <)$. For the latter domain, this contrasts with known undecidability results of (unbounded) register CLTL synthesis, by Bhaskar and Praveen. Lastly, we consider synthesis in a partial observation setting by extending CLTL with invisible variables.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Modal and temporal logics

Keywords and phrases Synthesis, Data words, Constraint linear time logic, Register transducer

Digital Object Identifier 10.4230/LIPIcs.CSL.2026.8

Funding *Emmanuel Filiot*: E. Filiot is research director at F.R.S.-FNRS. This work was partially funded by the FNRS/FWO Weave project T011724F.
Pierre-Alain Reynier: This work was partially funded by ANR project QuaSy (ANR-23-CE48-0008).

1 Introduction

Boolean reactive synthesis. Program synthesis – the automatic generation of programs from high-level specifications – has emerged as a promising avenue to improve software reliability by producing programs that are correct *by construction*.

A particularly rich and well-studied area within the field of program synthesis is *reactive synthesis* [3], which focuses on the design of algorithmic methods for the automatic construction of reactive systems, i.e. systems that maintain ongoing interactions with their environments. Over the past fifteen years, there has been remarkable progress in the synthesis of reactive systems modeled as finite state machines (Mealy machines), from specifications expressed in linear-time temporal logic (LTL). These developments have led to the creation of powerful tools and solvers, whose performance is continuously evaluated through the annual Reactive Synthesis Competition [15].

Beyond the Booleans. Classical reactive synthesis methods focus on complex control aspects and typically assume that reactive systems process a finite set of Boolean signals, while ignoring *data*. More recent research has considered extensions of this purely Boolean setting to *data-aware* reactive systems. In particular, there is a line of work on the synthesis of infinite-state systems modeled as transducers with registers. In this setting, programs targeted by synthesis algorithms are modeled as a Mealy machine extended with registers

that allow storing and comparing incoming data to produce data. In turn, the synthesis problem is parameterized by a *data domain*, i.e. an infinite set of data values and a finite set of predicates over those data values.

A natural candidate for expressing temporal specifications over data domains is *constraint LTL* (CLTL), which extends LTL with constraints over data values [9]. CLTL formulas are built over a finite set of variables X, which is extended by considering for any variable $x \in X$, $k+1$ copies $x^{(0)}, \ldots, x^{(k)}$ of x. The intended meaning of $x^{(i)}$ is to denote the content of x in i steps. The syntax of CLTL formulas is defined as for LTL, except that atomic formulas are predicates over the (copied) variables. As an example, the CLTL formula $G(x^{(0)} = y^{(2)})$ is satisfied if, at any instant t, x holds the same data value as y at instant $t+2$. Over a set of data values $\mathbb{D}$, models of CLTL formulas are sequences of valuations $(X \to \mathbb{D})^\omega$. It is worth noting that the satisfiability of a CLTL formula depends on the data domain. For instance, $G(x^{(0)} > x^{(1)})$ is satisfiable in $\mathbb{Z}$ but not in $\mathbb{N}$. The formula $G(x^0 < x^{(1)} \leq y^{(0)} = y^{(1)})$ is satisfiable in $\mathbb{Q}$ but not in $\mathbb{Z}$.

In an attempt to classify data domains, the *completion property* has been considered in the literature [9]. For a data domain $\mathcal{D}$, it asks that for any satisfiable constraint C (defined as a conjunction of predicates over $\mathcal{D}$), any partial valuation satisfying a sub-constraint of C can be extended to a (total) valuation satisfying C. For example, the constraint $C_0 = x < y < z$ is satisfiable in $\mathbb{Q}$, and any valuation of x, z satisfying $x < z$ can be extended to a valuation of x, y, z satisfying C_0, e.g. by taking $y = (x+z)/2$. While any domain $(\mathbb{D}, =)$ and the domain $(\mathbb{Q}, <)$ satisfy the completion property, important data domains such as $(\mathbb{N}, <)$ and $(\mathbb{Z}, <)$ do not. As an example, the partial valuation $x = 0, z = 1$, which satisfies $x < z$, cannot be extended to a valuation that satisfies C_0.

While the *satisfiability* problem for CLTL (over various data domains) has been quite studied in the literature [5, 9] (see also the survey [7] and the references therein), not much is known about the *synthesis* problem. In this context, the set of variables is further divided into variables $\mathbb{I}$ controlled by the environment (input variables) and variables $\mathbb{O}$ controlled by the system (output variables). The synthesis problem from specifications expressed in CLTL has been studied in [2]. It consists in deciding whether there exists a strategy that, at each step, reads data values of $\mathbb{I}$, and outputs data values for $\mathbb{O}$, resulting in an infinite data word over $\mathbb{I} \cup \mathbb{O}$ that satisfies the CLTL formula. It is shown that over data domains with the completion property, CLTL synthesis is decidable. However, synthesis is shown to be *undecidable* for $(\mathbb{Z}, <)$. To recover decidability, the authors of [2] consider some restriction, called single-sidedness, in which the power of the environment is restricted. In this paper, we take an orthogonal route to recover decidability.

Register-bounded reactive synthesis. While [2] considers arbitrary strategies, we focus on strategies represented by *register transducers*. A register transducer successively receives as input a valuation $(\mathbb{I} \to \mathcal{D})$, compares it with the data values stored in its registers, assigns some of the input values to its registers (or none), and finally produces some output valuation $(\mathbb{O} \to \mathcal{D})$ by assigning to each variable in $\mathbb{O}$ the content of some register. These strategies are more amenable to implementations, as they are close to real programs. We consider the *register-bounded synthesis problem from* CLTL, defined as the problem of deciding, given as input a CLTL formula Φ over $\mathcal{D}$ and some integer r, whether there exists a transducer with r registers which realizes Φ (and in that case output it). As we show, bounding the number of registers (but not the number of states) allows to recover decidability for CLTL synthesis over data domains such as $(\mathbb{Z}, <)$. Register-bounded synthesis is also desirable for synthesizing systems with a *minimal* number of registers. As registers represent some auxiliary memory, this is particularly relevant in contexts where resources are limited.

Contributions. Our objective is to develop general techniques that allow us to show decidability results for *families* of data domains. To that end, we develop reductions from the register-bounded synthesis problem to a realizability problem over a finite alphabet. The key in this approach is the analysis of the set $\mathcal{S}_{\mathcal{D}}$ of infinite constraint sequences over $\mathcal{D}$ that are satisfiable. Our contributions are as follows.

1. We first show that if $\mathcal{S}_{\mathcal{D}}$ is an ω-regular language, then the register-bounded synthesis problem from CLTL is decidable (Theorem 10). In particular, if a data domain $\mathcal{D}$ meets the completion property, then $\mathcal{S}_{\mathcal{D}}$ is ω-regular. This allows us to prove that the register-bounded synthesis problem from CLTL is 2ExpTime-c over any domain $(\mathbb{D}, =)$ for an infinite set $\mathbb{D}$, and over the domain $(\mathbb{Q}, <)$ (Corollary 19).
2. Yet, $\mathcal{S}_{(\mathbb{Z},<)}$ is not ω-regular (see [10]), which reflects the undecidability result mentioned earlier. As a remedy, we show that if $\mathcal{S}_{\mathcal{D}}$ can be over-approximated by an ω-regular language which coincides with $\mathcal{S}_{\mathcal{D}}$ on lasso words, then the register-bounded synthesis problem from CLTL is decidable (Theorem 22). We show that this is the case for numerous data domains including $(\mathbb{N}, <)$, $(\mathbb{Z}, <)$, $(\Sigma^*, \preceq_{pref})$ for Σ any finite alphabet and $\preceq_{pref}$ the prefix relation, as well as $(\mathbb{Z}^k, <_k)$ for any k and $<_k$ the partial order on tuples (pairwise comparison), using a reduction to [16]. This implies that the register-bounded synthesis problem from CLTL is 2ExpTime-c for all these data domains (Corollary 28).
3. Lastly, we consider a partial-observation generalization in which the CLTL formula has private and public input variables, but the system only has access to public ones. We show that our approach can be leveraged to this setting, yielding that the problem is 2ExpTime-c for any domain $(\mathbb{D}, =)$, and over the domain $(\mathbb{Q}, <)$ (Corollary 32).

Related work. Synthesis problems of register transducers over data domains have already been considered in the literature, but mostly when the specification is already given as an automaton, and in particular a *register automaton*, see [11, 12, 17, 18]. Beyond specifications given by deterministic register automata, synthesis quickly turns into undecidability. That is the case, for instance, for specifications given by non-deterministic or universal register automata [11, 17], which are strictly more expressive than their deterministic restrictions. When considering the data domains $(\mathbb{Z}, <)$ and $(\mathbb{N}, <)$, synthesis is already undecidable for specifications given by deterministic register automata [10].

Register-bounded synthesis has been considered for specifications given as register automata [11, 16, 17]. In particular, for specifications given as universal register automata over $(\mathbb{Z}, =)$ and $(\mathbb{Z}, <)$, register-bounded synthesis has been shown to be decidable [12, 16, 17]. A legitimate question is whether register-bounded synthesis from CLTL could be reduced to register-bounded synthesis from register automata and directly apply the results of [12, 16, 17]. We do not know of any reduction that would preserve the number of registers needed to realize the specification. Indeed, even though CLTL formulas can be converted into deterministic register automata, this conversion is however modulo some model encoding: CLTL formulas are interpreted over $(X \to D)^\omega$ while register automata are executed on words in D^ω, and so tuples need to be encoded as chunks of $d = |X|$ consecutive data. Having access to d data at once allows for register succinctness. This is because register transducers in [12, 16, 17] have access to only one data at a time, while the register transducers of this paper receive as input a tuple of data, to match the semantics of CLTL formulas. For example, a CLTL specification might require to output an input value only if it appears twice in a tuple. To check the latter property, a register transducer in [12, 16, 17] would need $d-1$ additional registers.

On top of the related papers already cited before, we would like to mention the work on *constraint (infinite) tree automata* over $(\mathbb{Z}, <)$, whose emptiness problem is proved to be decidable in [8]. Whereas the connection between tree automata and synthesis is well-known in the boolean setting (see for instance [20]), it is not clear how register-bounded synthesis from constraint LTL could be reduced to the emptiness problem of the constraint tree automata of [8].

2 Constraint LTL and automata over data words

Finite and infinite words. Given an alphabet Σ finite or not, let Σ^* (resp. Σ^ω) be the set of finite (resp. infinite) sequences, called words, over Σ. In this paper, we denote finite words as $\overline{u} = u_1 u_2 \dots u_n$ where $u_i \in \Sigma$ for all $i \in \{1, \dots, n\}$. We let $|\overline{u}| = n$ denote its length and for $1 \leq i \leq j \leq |\overline{u}|$, we let $\overline{u}_{[i,j]} = u_i u_{i+1} \dots u_j$. Similarly, if $\overline{u} = u_1 u_2 \dots$ is an infinite word, we let $\overline{u}_{[i,+\infty]} = u_i u_{i+1} \dots$ be the infinite suffix starting at position i.

Data. We define a data domain $\mathcal{D}$ as a tuple $(\mathbb{D}, P, d_0)$, where $\mathbb{D}$ is an infinite set whose elements are called *data*, P is a finite set of predicates (relations over $\mathbb{D}$ of any arity) and $d_0 \in \mathbb{D}$ is a constant. It is also required that the set of predicates always contains the equality predicate, in order to be able to distinguish data values. In this paper, we will mostly use the data domains $(\mathbb{D}, =, d_0)$ (where $\mathbb{D}$ is arbitrary), $(\mathbb{Q}, <, 0)$ and $(\mathbb{N}, <, 0)$. Note that $(\mathbb{N}, <, 0)$ does not contain equality, but $x = y$ can be expressed as $\neg(x < y) \wedge \neg(y < x)$. In general, data domains are defined as interpretations of a set of predicate *symbols*; however, to simplify the presentation, in this paper we confuse symbols with their (intuitive) semantics. Lastly, we say that a data domain is *decidable* if its existential first-order theory is decidable.

Constraints. Let V be a finite set of variables, and let $\mathcal{D} = (\mathbb{D}, P, d_0)$ be a data domain. A *valuation* from V to $\mathbb{D}$ is a (total) mapping ν from V to $\mathbb{D}$. We extend it to $V \cup \{d_0\}$ by letting $\nu(d_0) = d_0$. We denote by $Val_{V,\mathbb{D}}$ the set of valuations from V to $\mathbb{D}$. Given two valuations ν_1, ν_2 defined over two distinct subsets A_1, A_2 of V, we let $\nu_1 \uplus \nu_2$ be the valuation defined on $A_1 \cup A_2$ by $\nu_1 \uplus \nu_2(x) = \nu_i(x)$ if $x \in A_i$.

A literal ℓ over V is a term of the form $p(x_1, \dots, x_k)$ or its negation $\neg p(x_1, \dots, x_k)$, with $p \in P$ of arity k and $x_1, \dots, x_k \in V \cup \{d_0\}$. It is satisfied by a valuation $w \in Val_{V,\mathbb{D}}$, written $w \models \ell$, if $(w(x_1), \dots, w(x_k)) \in p$ if $\ell = p(x_1, \dots, x_k)$ (and $(w(x_1), \dots, w(x_k)) \notin p$ for a negative literal). A *constraint* C over V is a conjunction of literals over V. We often view C as the set of its conjuncts. We denote by $\mathcal{C}_V$ the set[1] of constraints over V. We say that a constraint $C \in \mathcal{C}_V$ is *satisfiable* if there exists a valuation $w \in Val_{V,\mathbb{D}}$ such that $w \models \ell$ for all literals $\ell \in C$. We write $w \models C$ when this holds.

A constraint C is *consistent* if it does not contain a literal and its negation. It is *maximally consistent* if it is consistent and for all predicates p of arity k and for all $(x_1, \dots, x_k) \in (V \cup \{d_0\})^k$, either $p(x_1, \dots, x_k)$ or its negation occurs in C. For example over $(\mathbb{N}, <, 0)$, the constraint $C_0 = (x < y) \wedge (0 < x) \wedge (0 < y) \wedge \neg(y < x) \wedge \neg(x < 0) \wedge \neg(y < 0)$ is not maximally consistent, but $C_1 = C_0 \wedge \neg(x < x) \wedge \neg(y < y) \wedge \neg(0 < 0)$ is. We denote by $\mathcal{MC}_V$ the set of maximally consistent constraints over V.

[1] Note that constraints may also use the constant symbol d_0. This is left implicit in the notation $\mathcal{C}_V$, for the sake of readability.

Completion property. For $V' \subseteq V$, we define the restriction $C|_{V'}$ of a constraint C, as the subset of literals of C that use only variables in V'. A satisfiable constraint C is *completable* if for any $V' \subseteq V$, for all $w' \in Val_{V',\mathbb{D}}$ such that $w' \models C|_{V'}$, there exists $w \in Val_{V,\mathbb{D}}$ such that $w'(x) = w(x)$ for all $x \in V'$ and $w \models C$.

A data domain has *completion property* (we also say it is *completable*) if all satisfiable and maximally consistent constraints are completable. For example, $(\mathbb{N}, <, 0)$ does not meet the completion property: if we consider the constraint C_1 defined before, it is satisfiable and maximally consistent. However, the valuation w' such that $w'(y) = 1$ satisfies $C_1|_{\{y\}}$, but we cannot find a value $d \in \mathbb{N}$ to map to x such that $(0 < d) \wedge (d < w'(y))$. However, a slight generalization of Corollary 5.4 in [9] to handle constants yields:

▶ **Lemma 1** ([9]). *Data domains* $(\mathbb{D}, =, d_0)$ *and* $(\mathbb{Q}, <, 0)$ *are completable.*

Data valuation words. Let V be a finite set of variables and $\mathcal{D} = (\mathbb{D}, P, d_0)$ be a data domain. A *data valuation word* is a sequence $\overline{w} = w_1 w_2 \cdots \in (Val_{V,\mathbb{D}})^\omega$ where each w_i is a valuation. Data valuation words are used to model the traces of the systems that we want to synthesize. As CLTL allows us to compare valuations that occur at different time points, we also introduce another notion of valuation word. Let $k \in \mathbb{N}$, and define the set of k-future variables as $V^{(k)} = \{x^{(i)} \mid x \in V, 0 \leq i \leq k\}$. A k-extended valuation word is a sequence $\overline{\nu} = \nu_1 \nu_2 \cdots \in (Val_{V^{(k)},\mathbb{D}})^\omega$. Intuitively, $\nu_i(x^{(j)})$ is intended to be the value of x at time $i+j$, and therefore it is required that $\nu_i(x^{(j)}) = \nu_{i+j}(x)$. This condition is enforced by the notion of *compatible* extended valuation words, defined as the sequences $\overline{\nu} \in (Val_{V^{(k)},\mathbb{D}})^\omega$ such that

$$\forall i > 0, \forall 0 < j \leq k, \forall x \in V, \nu_i(x^{(j)}) = \nu_{i+j}(x)$$

We define a bijection between valuation words and their compatible k-extended form, *via* the function $\textsc{extend}_k : (Val_{V,\mathbb{D}})^\omega \to (Val_{V^{(k)},\mathbb{D}})^\omega$ defined by $\textsc{extend}_k(\overline{w}) = \overline{\nu}$ where for all $j \geq 1$ and all $x^{(i)} \in V^{(k)}$, $\nu_j(x^{(i)}) = w_{j+i}(x)$. One can show that $\textsc{extend}_k$ is a bijection between valuation words and compatible k-extended valuation words, and we denote by $\textsc{compress}$ its inverse. Last, we lift $\uplus$ from valuations to valuation words in the obvious way.

▶ **Example 2.** Let $V = \{x, y\}$ and $\mathcal{D} = (\mathbb{N}, <, 0)$. We consider $\overline{w} \in (Val_{V,\mathbb{D}})^\omega$ given by $w_i(x) = i$ and $w_i(y) = i + 1$ for all $i \geq 1$. Then $\overline{\nu} = \textsc{extend}_1(\overline{w})$ is given by $\nu_i(x^{(0)}) = i, \nu_i(x^{(1)}) = \nu_i(y^{(0)}) = i + 1, \nu_i(y^{(1)}) = i + 2$ for all $i \geq 1$.

Constraint words. A *constraint word* is a sequence $\overline{c} = C_1 C_2 \cdots \in (\mathcal{C}_V)^\omega$ of constraints over V. A valuation word $\overline{w} \in (Val_{V,\mathbb{D}})^\omega$ satisfies $\overline{c}$, written $\overline{w} \models \overline{c}$, if for all $i \geq 1, w_i \models C_i$. We let $[\![\overline{c}]\!] = \{\overline{w} \in (Val_{V,\mathbb{D}})^\omega \mid \overline{w} \models \overline{c}\}$. Given $\overline{c'} \in (\mathcal{C}_V)^\omega$, we write $\overline{c} \subseteq \overline{c'}$ whenever for all $i \geq 1$, $C_i \subseteq C'_i$ (seen as sets of literals). In particular, $\overline{c} \subseteq \overline{c'}$ implies $[\![\overline{c'}]\!] \subseteq [\![\overline{c}]\!]$.

For clarity purposes, we write $\mathcal{C}_V^{(k)}$ for the set of constraints over $V^{(k)}$ instead of $\mathcal{C}_{V^{(k)}}$, the same goes for $\mathcal{MC}_V^{(k)}$. Constraint words in $\mathcal{C}_V^{(k)}$ are said to be of *rank* k. The satisfiability notion is extended to constraint words $\overline{c} \in (\mathcal{C}_V^{(k)})^\omega$ of rank k via the function $\textsc{extend}_k$. Formally, $\overline{w} \in (Val_{V,\mathbb{D}})^\omega$ satisfies $\overline{c}$ if $\textsc{extend}_k(\overline{w})$ satisfies $\overline{c}$, and $[\![\overline{c}]\!] = \{\overline{w} \in (Val_{V,\mathbb{D}})^\omega \mid \overline{w} \models \overline{c}\}$.

CLTL. Fix a data domain $\mathcal{D} = (\mathbb{D}, P, d_0)$ and a finite set of variables V. $\mathsf{CLTL}(\mathcal{D})$-formulas over V are defined inductively as follows. The atomic formulas of rank $k \geq 0$ are terms of the form $p(x_1, \ldots, x_\ell)$, where $p \in P$ is a predicate of arity ℓ and $(x_1, \ldots, x_\ell) \in (V^{(k)} \cup \{d_0\})^\ell$. We denote this set by $Atom_k$. CLTL-formulas of rank k are inductively defined as:

$$\Phi ::= \Phi_1 U \Phi_2 \mid \Phi_1 \Rightarrow \Phi_2 \mid \neg\Phi \mid X\Phi \mid a \in Atom_k.$$

A CLTL-formula is a CLTL-formula of rank k for some k. As usual, the Boolean connectives $\wedge$ and $\vee$ can be defined from $\Rightarrow$ and $\neg$, and we let $\top = a \vee \neg a$ (for some atomic formula a) and $\bot = \neg\top$. Moreover, as for LTL formulas, we define the temporal operators *globally* (G) and *eventually* (F) by $F\Phi = \top U \Phi$ and $G\Phi = \neg F \neg \Phi$. Finally, we may write x instead of $x^{(0)}$, and just CLTL instead of CLTL when $\mathcal{D}$ is irrelevant.

The *size* $|\Phi|$ of a formula Φ is defined as the number of symbols in Φ, with the exception of variables $x^{(i)}$ that are defined to be of size i.

Semantics. We define the semantics of CLTL by induction on formulas. Let Φ be a CLTL formula over V of rank k, and $\overline{\nu} \in (Val_{V^{(k)},\mathbb{D}})^\omega$. We say that $\overline{\nu}$ satisfies Φ, written $\overline{\nu} \models \Phi$, if the following holds (inductively):

- if $\Phi = p(\vec{x})$ then $\nu_1 \models p(\vec{x})$
- if $\Phi = \neg\Psi$ then $\overline{\nu} \not\models \Psi$
- if $\Phi = \Psi_1 \Rightarrow \Psi_2$ then $\overline{\nu} \not\models \Psi_1$ or $\overline{\nu} \models \Psi_2$
- if $\Phi = X\Psi$ then $\nu_{[2,+\infty]} \models \Psi$
- if $\Phi = \Psi_1 U \Psi_2$ then $\exists n \geq 1$ such that $\forall 1 \leq i < n$, $\overline{\nu}_{[i,+\infty]} \models \Psi_1$ and $\overline{\nu}_{[n,+\infty]} \models \Psi_2$

Given a valuation word $\overline{w} \in (Val_{V,\mathbb{D}})^\omega$, we say that $\overline{w}$ *satisfies* Φ, also written $\overline{w} \models \Phi$, if $\textsc{extend}_k(\overline{w}) \models \Phi$ holds. We let $L(\Phi) = \{\overline{w} \in (Val_{V,\mathbb{D}})^\omega \mid \overline{w} \models \Phi\}$.

▶ **Example 3** (Example 2 continued)**.** Consider $\Phi = G\left(x \leq y \wedge y \leq y^{(1)}\right) \in$ CLTL. One can easily check that $\overline{w} \in [\![\Phi]\!]$ holds. The same holds for $G(y < x^{(2)} \wedge \neg(y < x^{(1)}))$.

▶ **Definition 4.** *A constraint automaton (CA) $\mathcal{A}$ over $\mathcal{D}$ is a tuple (Q, Δ, I, χ, k) where Q is a finite set of states, $k \in \mathbb{N}$ is the maximal rank for the constraints, $\Delta \subseteq Q \times \mathcal{C}_V^{(k)} \times Q$ is a transition relation, $I \subseteq Q$ is a set of initial states and $\chi \in Val_{Q,\mathbb{N}}$ a coloring of the states.*

A run of a CA over a valuation word $\overline{w} \in (Val_{V,\mathbb{D}})^\omega$ is a sequence of transitions $\overline{t} \in \Delta^\omega$ such that for all $i \geq 1$, $t_i = (q_{i-1}, C_i, q_i)$, with $w_i \models C_i$ and $q_0 \in I$. A run is *accepting* if the highest color seen infinitely often along the run is even. Under the non-deterministic semantics, the word $\overline{w}$ is accepted if there exists an accepting run. This yields the model of non-deterministic parity CA. The number of priorities of $\mathcal{A}$ is the size of $\chi(Q)$. We will also consider a universal semantics: in this case, a word $\overline{w}$ is accepted if *all* runs on $\overline{w}$ are accepting. An automaton is *deterministic* if for any two transitions of the form (q, C_1, p_1) and (q, C_2, p_2) such that $p_1 \neq p_2$, for all valuation $w \in Val_{V,\mathbb{D}}$, either $w \not\models C_1$ or $w \not\models C_2$. We also consider subclasses of CA: we say that an automaton has a Büchi acceptance condition (resp co-Büchi) if all its states have color 1 or 2, that is, $\chi(Q) \subseteq \{1, 2\}$ (resp $\chi(Q) \subseteq \{0, 1\}$). It has a reachability (resp safety) acceptance condition if it has a Büchi (resp co-Büchi) acceptance condition and there exists no transition from a state colored 2 to a 1 (resp from 1 to 0). Last, the language of a CA $\mathcal{A}$ is the set of accepted words, and is denoted $L(\mathcal{A})$.

We call *syntactic automaton* $\mathcal{A}_{stx}$ the automaton $\mathcal{A}$ seen as a finite automaton over the finite alphabet $\mathcal{C}_V^{(k)}$ of constraints of rank k and let $L_{stx}(\mathcal{A}) = L(\mathcal{A}_{stx})$ denote its language.

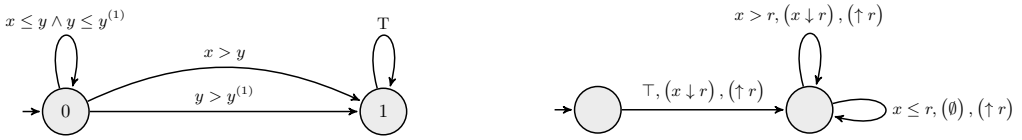
From CLTL to constraint automata. It is well-known any LTL sentence can be converted into a Büchi automaton. This result carries over to CLTL and constraint automata.

▶ **Proposition 5.** *Let Φ be a* CLTL *formula. One can construct in* ExpTime *a universal co-Büchi CA $\mathcal{A}$ such that $L(\Phi) = L(\mathcal{A})$ whose size is exponential in the size of Φ.*

Proof. The logic CLTL can be seen as LTL over the finite alphabet $\mathcal{C}_V^{(k)}$. As such we can use the classical exponential procedure that converts any LTL formula into an equivalent non-deterministic Büchi automaton, to build a non-deterministic Büchi constraint automaton

equivalent to the CLTL formula (see Section 3.2 in [7]). The result follows by applying the latter construction on the negation of the formula and interpreting the resulting automaton with a universal co-Büchi condition. ◀

▶ **Example 6** (Example 3 continued). A constraint automaton equivalent to the CLTL formula Φ of Example 3 is depicted in Figure 1a.



(a) A universal co-Büchi CA. The colors are indicated in the states.

(b) A register transducer.

Figure 1 A constraint automaton and a register transducer.

3 Register-bounded synthesis problem from CLTL

3.1 Synthesis problem over finite alphabets

We first recall the synthesis problem for ω-regular specifications over a finite alphabet. In this setting, the goal is to synthesize (if it exists) a finite transducer over a finite input alphabet Σ_{in} and a finite output alphabet Σ_{out} that realizes a specification $S \subseteq (\Sigma_{in}\Sigma_{out})^\omega$, given, for example, as a non-deterministic Büchi automaton. Let us formally define this problem. A *finite transducer* (FT for short) is a tuple $T = (\Sigma_{in}, \Sigma_{out}, Q, q_0, \delta)$ such that Q is a finite set of states with initial state $q_0 \in Q$, and $\delta : Q \times \Sigma_{in} \to \Sigma_{out} \times Q$ is a (total) transition function. The semantics of T is a language, denoted $L(T) \subseteq (\Sigma_{in}\Sigma_{out})^\omega$, defined as the set of words $i_1 o_1 i_2 o_2 \cdots \in (\Sigma_{in}\Sigma_{out})^\omega$ such that there exists a sequence of states $p_0 p_1 \cdots \in Q^\omega$ with $p_0 = q_0$ and for all $j \geq 1$, $\delta(q_j, i_j) = (o_j, q_{j+1})$.

A specification $S \subseteq (\Sigma_{in}\Sigma_{out})^\omega$ is said to be *realizable* if there exists a finite transducer T such that $L(T) \subseteq S$. Büchi-Landweber's Theorem states that when S is ω-regular, the realizability problem is decidable [4]. More precisely, this problem can be solved in ExpTime when S is given as a universal co-büchi automaton [20], and is 2ExpTime-c when S is given as an LTL formula over some sets of input and output atomic propositions [21].

3.2 Register transducers over data words

We now extend synthesis to infinite alphabets and CLTL specifications. We need to adapt the model of implementation, *i.e.* transducers. To this end, we first extend the notion of finite transducer to transducers over infinite alphabets that act over a finite set of registers, formally defined via the notion of *actions* below.

Action words. Let $\mathcal{D} = (\mathbb{D}, P, d_0)$ be a data domain and $V = \mathbb{I} \uplus \mathbb{O}$ be a finite set of variables partitioned into $\mathbb{I}$ (input variables) and $\mathbb{O}$ (output variables). They are also called system and environment variables in the context of synthesis. Let R be a finite set of elements called registers. An *action* over R and V is a tuple in $Test_{\mathbb{I},R} \times Assign_{\mathbb{I},R} \times Output_{\mathbb{O},R}$ where:

- $Test_{\mathbb{I},R} = \mathcal{MC}_{\mathbb{I} \cup R}$ is the set of maximally consistent constraints over $\mathbb{I} \cup R$.
- $Assign_{\mathbb{I},R}$ is the set of partial functions $\rho^{ass} : R \hookrightarrow \mathbb{I}$ from R to $\mathbb{I}$.
- $Output_{\mathbb{O},R} = Val_{\mathbb{O},R}$ is the set of (total) functions $\rho^{out} : \mathbb{O} \to R$ from $\mathbb{O}$ to R.

Assignments are to be understood as a way to store data received as input, to be able to compare them later to new incoming data, or to output them. The output functions specify for each variable, which register should be used to obtain the value that must be produced at each step. We denote by $Act_{\mathbb{I},\mathbb{O},R}$ the set of actions. An *action word* is an infinite word $\overline{a} \in (Act_{\mathbb{I},\mathbb{O},R})^\omega$. We denote by $(Act_{\mathbb{I},\mathbb{O},R})^\omega$ the set of action words.

The semantics of an action word $\overline{a} = (C_1, \rho_1^{ass}, \rho_1^{out})(C_2, \rho_2^{ass}, \rho_2^{out})\dots$ is a language $[\![\overline{a}]\!] \subseteq (Val_{V,\mathbb{D}})^\omega$ that we now define. Assume $\overline{w} = (w_1^V = w_1^{\mathbb{I}} \uplus w_1^{\mathbb{O}})(w_2^V = w_2^{\mathbb{I}} \uplus w_2^{\mathbb{O}})\dots$ be a sequence of valuations. We "execute" the action word $\overline{a}$ on $\overline{w}$. Registers are used to store data from the input valuations in $\overline{w}$, and the actions on registers (tests and assignments) are dictated by $\overline{a}$. Registers in R are all initialized with the value d_0 (valuation w_1^R). Then, test C_1 is performed on $w_1^{\mathbb{I}} \cup w_1^R$. If it succeeds, registers are updated according to ρ_1^{ass} (giving valuation w_2^R), and the valuation $w_1^{\mathbb{O}} = w_2^R \circ \rho_1^{out}$ is output. The execution proceeds with the new action $(C_2, \rho_2^{ass}, \rho_2^{out})$ and register valuation w_2^R. This is formally captured by the following compatibility notion.

We say that a valuation word $\overline{w} = (w_1 = w_1^{\mathbb{I}} \uplus w_1^{\mathbb{O}})(w_2 = w_2^{\mathbb{I}} \uplus w_2^{\mathbb{O}}) \cdots \in (Val_{V,\mathbb{D}})^\omega$ is *compatible with an action word* $\overline{a} \in (Act_{\mathbb{I},\mathbb{O},R})^\omega$ if there exists a register valuation word $\overline{w^R} = w_1^R w_2^R \cdots \in (Val_{R,\mathbb{D}})^\omega$ such that $w_1^R(r) = d_0$ for all $r \in R$ and for all $i \in \mathbb{N}_{>0}$:

- $w_i^{\mathbb{I}} \uplus w_i^R \models C_i$;
- $w_{i+1}^R(r) = w_i^{\mathbb{I}} \circ \rho_i^{ass}(r)$ if $\rho_i^{ass}(r)$ is defined, otherwise $w_{i+1}^R(r) = w_i^R(r)$;
- $w_i^{\mathbb{O}} = w_{i+1}^R \circ \rho_i^{out}$.

The *language of an action word* $\overline{a}$ is the set $[\![\overline{a}]\!] \subseteq (Val_{V,\mathbb{D}})^\omega$ of valuation words compatible with $\overline{a}$. We write $\overline{w} \models \overline{a}$ whenever $\overline{w} \in [\![\overline{a}]\!]$. It is worth observing that if $\overline{w} \in [\![\overline{a}]\!]$, then the associated register valuation word $\overline{w^R}$ is unique.

▶ **Definition 7** (Register transducer). *A register transducer (RT for short) over $\mathcal{D}$ is a tuple $\mathbb{T} = (Q, q_0, R, V, \delta)$ where:*

- *R is a finite set of registers,*
- *$V = \mathbb{I} \uplus \mathbb{O}$ is a finite set of variables partitioned into input variables and output variables such that $V \cap R = \varnothing$,*
- *$(Test_{\mathbb{I},R}, Assign_{\mathbb{I},R} \times Output_{\mathbb{O},R}, Q, q_0, \delta)$ is a finite transducer, denoted $\mathbb{T}_{stx}$.*

Observe that, as we consider tests defined by maximally consistent constraints, the transducers we consider are deterministic and complete w.r.t. their input.

We define two semantics for T. The first is more syntactic and is useful to define finite abstractions of languages defined by register transducers, and is simply defined as $L_{stx}(\mathbb{T}) = L(\mathbb{T}_{stx})$. The second one, over valuations of V, is defined as $L(\mathbb{T}) = \bigcup_{\overline{a} \in L_{stx}(\mathbb{T})} [\![\overline{a}]\!]$.

3.3 Synthesis problem over data domains

When moving from a finite alphabet to data words over some data domain $\mathcal{D}$, we lift specifications from ω-regular languages over a finite alphabet to languages $L \subseteq (Val_{V,\mathbb{D}})^\omega$, for some set of variables V. Such a specification can, of course, be described by a constraint automaton, as well as by a CLTL formula, which is the purpose of this work. Regarding implementations, we consider register transducers, leading to the following problem:

Register-bounded Synthesis Problem from CLTL
Input: A CLTL($\mathcal{D}$) formula Φ over $V = \mathbb{I} \uplus \mathbb{O}$ and an integer r
Output: A register transducer T over V and with r registers, if it exists, which *realizes* Φ, i.e. such that $L(T) \subseteq L(\Phi)$.

The decision problem associated with the synthesis problem is called *the realizability problem*, and only asks whether such a transducer T exists. In this paper, when stating hardness results with respect to the complexity of the synthesis problem, they refer to the realizability problem. Moreover, when stating upper-bounds, they also include the cost of constructing a solution (a register transducer).

We assume that r is given in unary. This is reasonable as the expected register transducer has r registers, which means that the configuration of the registers is already of size r.

▶ **Example 8** (Example 3 continued)**.** We consider again $\Phi = G\left(x \leq y \wedge y \leq y^{(1)}\right)$ with $\mathbb{I} = \{x\}$ and $\mathbb{O} = \{y\}$. In our context, since RT can only output data they received before, a transducer realizing this specification must output the largest input seen so far. The RT with a single register depicted in Figure 1b realizes this specification. The transitions read as follows: $x \downarrow r$ means that the value of x is assigned to register r, and $\uparrow r$ means that the content of r is output. Observe that the tests on the transitions of this RT are not maximally consistent. This is just a graphical simplification for the sake of conciseness.

4 Data domains with ω-regular satisfiability

Let $\mathcal{D}$ be a data domain. For all integer $k \geq 1$ and set of variables X, let

$$\mathsf{SAT}_{k,X} = \{\overline{c'} \in (\mathcal{C}_X^{(k)})^\omega \mid \overline{c'} \text{ is a word of maximally consistent constraints and } [\![\overline{c'}]\!] \neq \emptyset\}$$

be the set of satisfiable constraint words in $\mathcal{D}$.

▶ **Definition 9.** *We say that $\mathcal{D}$ has* effective ω-regular satisfiability *if the function which given some k and X returns an automaton recognizing $\mathsf{SAT}_{k,X}$, is computable.*

In this section, we will show the following theorem.

▶ **Theorem 10.** *Let $\mathcal{D}$ be a data domain with effective ω-regular satisfiability. Then register-bounded synthesis from specifications expressed as universal co-Büchi CA over $\mathcal{D}$ is decidable.*

The main idea is to reduce the problem to a synthesis problem for an ω-regular specification over a finite alphabet, using some sound and complete finite abstraction stated in Lemma 13. Since the goal is to synthesize a register transducer, the finite alphabet is the set of actions $Act_{\mathbb{I},\mathbb{O},R}$ of the transducer, which is partitioned into input actions (constraints over $\mathbb{I}$ and R) and output actions (register assignments and output function). The next two lemmas are technical results towards proving Lemma 13. From now on, we fix $\mathcal{D}$ some data domain.

From action words to constraint words. First, in order to compare actions of register transducers and constraints of constraint automata, sequences of actions are canonically mapped to sequences of constraints, via a mapping $\mathsf{cstr} : (Act_{\mathbb{I},\mathbb{O},R})^\omega \to (\mathcal{C}_{V\cup R}^{(1)})^\omega$. Remember that an action is a triple $(C, \rho^{ass}, \rho^{out})$ where C is a constraint over $\mathbb{I} \cup R$, $\rho^{ass} : R \hookrightarrow \mathbb{I}$ is a partial function and $\rho^{out} : \mathbb{O} \to R$ a function. The latter two assignments induce constraints between V and R. It is also worth noting that the register valuation at step i is the one used for the test, while the one at step $i+1$ is the one obtained after the assignment, which is used for producing the output. For example, the assignment ρ^{ass} implies that the *next* content of register r is equal to the value of variable $\rho^{ass}(r)$. Similarly, any output variable $y \in \mathbb{O}$ holds a value equal to the *next* content of register $\rho^{out}(y)$. Formally, for all $\overline{a} = (C_1, \rho_1^{ass}, \rho_1^{out}) \ldots$, we let $\mathsf{cstr}(\overline{a}) = (C_1' \wedge init)C_2'C_3' \ldots$ where $init =_{def} \left(\bigwedge_{r\in R} r = d_0\right)$ and for all $i \geq 1$:

$$C_i' = C_i \wedge \bigwedge_{\substack{r \in R \text{ s.t.} \\ \rho_i^{ass}(r) \text{ defined}}} (r^{(1)} = \rho_i^{ass}(r)) \wedge \bigwedge_{\substack{r \in R \text{ s.t.} \\ \rho_i^{ass}(r) \text{ not defined}}} (r^{(1)} = r) \wedge \bigwedge_{y\in\mathbb{O}} (y = (\rho_i^{out}(y))^{(1)})$$

The latter transformation is correct in the following sense:

▶ **Lemma 11.** *Let* $\overline{a} \in (Act_{\mathbb{I},\mathbb{O},R})^\omega$ *then* $[\![\overline{a}]\!] = [\![\mathit{cstr}(\overline{a})]\!]|_V$.

Since a specification has constraints over $V^{(k)}$, and a register transducer expresses properties of action words, which themselves hold constraints over $V = \mathbb{I} \uplus \mathbb{O}$ and R, one needs a mechanism to synchronize these two types of constraints. This is done via the following function called *extension*, denoted $\mathsf{join} : (Act_{\mathbb{I},\mathbb{O},R})^\omega \times (\mathcal{C}_V^{(k)})^\omega \to \mathcal{MC}_{V\cup R}^{(k)}$, defined for any $\overline{a} \in (Act_{\mathbb{I},\mathbb{O},R})^\omega$ and $\overline{c} \in (\mathcal{C}_V^{(k)})^\omega$ as

$$\mathsf{join}(\overline{a}, \overline{c}) = \{\overline{c'} \in \mathcal{MC}_{V\cup R}^{(k)} \mid \mathsf{cstr}(\overline{a}) \subseteq \overline{c'} \text{ and } \overline{c} \subseteq \overline{c'}\}$$

Valuations satisfying constraint words in $\mathsf{join}(\overline{a}, \overline{c})$, when restricted to variables in V, satisfy both $\overline{a}$ and $\overline{c}$. Using Lemma 11, we prove:

▶ **Lemma 12** (Adequation)**.** *Let* $\overline{a} \in (Act_{\mathbb{I},\mathbb{O},R})^\omega$, $\overline{c} \in (\mathcal{C}_V^{(k)})^\omega$, $\overline{w} \in Val_{V\cup R,\mathbb{D}}$ *and* $\overline{c'} \in (\mathcal{MC}_{V\cup R}^{(k)})^\omega$ *such that* $\overline{w} \models \overline{c'}$. *Then* $\overline{c'} \in \mathit{join}(\overline{a}, \overline{c})$ *iff* $\overline{w}|_V \models \overline{a}$ *and* $\overline{w}|_V \models \overline{c}$.

The next lemma formalizes the reduction of synthesis from infinite to finite alphabets, and in particular to the alphabet of *actions* $Act_{\mathbb{I},\mathbb{O},R}$, which is finite as the set of registers R is finite. In turn, a specification given by a universal coBüchi CA $\mathcal{A}$ is translated into a specification $\mathsf{W}_{\mathcal{A},R}$ over the alphabet $Act_{\mathbb{I},\mathbb{O},R}$. A finite transducer realizing $\mathsf{W}_{\mathcal{A},R}$ can then be regarded as a transducer with set of registers R. The specification $\mathsf{W}_{\mathcal{A},R}$ is precisely defined to make sure that the latter register transducer satisfies $L(\mathcal{A})$. Informally, in the finite alphabet specification, an action word $\overline{a}$ is considered to be correct if any constraint word $\overline{c}$ such that both $\overline{a}$ and $\overline{c}$ are satisfiable by the *same* data valuation word, belongs to $L_{stx}(\mathcal{A})$. Formally, the reduction is ensured by the following transfer lemma:

▶ **Lemma 13** (Transfer Lemma)**.** *Let* $\mathcal{A}$ *be a universal co-Büchi CA over* $V = \mathbb{I} \uplus \mathbb{O}$ *and* R *a set of registers. Let*

$$\mathsf{W}_{\mathcal{A},R} = \{\overline{a} \in (Act_{\mathbb{I},\mathbb{O},R})^\omega \mid \forall \overline{c} \in (\mathcal{C}_V^{(k)})^\omega, (\exists \overline{c'} \in \mathit{join}(\overline{a}, \overline{c}), \overline{c'} \in \mathsf{SAT}_{k,V\cup R}) \Rightarrow \overline{c} \in L_{stx}(\mathcal{A})\}$$

$L(\mathcal{A})$ *is realizable by an RT with* $|R|$ *registers iff* $\mathsf{W}_{\mathcal{A},R}$ *is realizable by an FT.*

Proof. $\Rightarrow$ Suppose $\mathcal{A}$ is realized by some RT $\mathbb{T}$, i.e. $L(\mathbb{T}) \subseteq L(\mathcal{A})$. We show that $\mathbb{T}_{stx}$ realizes $\mathsf{W}_{\mathcal{A},R}$, i.e., $L(\mathbb{T}_{stx}) \subseteq \mathsf{W}_{\mathcal{A},R}$. Let $\overline{a} \in L(\mathbb{T}_{stx})$. Let $\overline{c} \in (\mathcal{C}_V^{(k)})^\omega$ such that $\exists \overline{c'} \in \mathsf{join}(\overline{a}, \overline{c})$ and $\overline{c'} \in \mathsf{SAT}_{k,V\cup R}$. By definition of $\mathsf{SAT}_{k,V\cup R}$, there exists $\overline{w} \in (Val_{V\cup R,\mathbb{D}})^\omega$, such that $\overline{w} \models \overline{c'}$. By Lemma 12, $\overline{w}|_V \models \overline{a}$ and $\overline{w}|_V \models \overline{c}$. As $\overline{a} \in L(\mathbb{T}_{stx})$ and $\overline{w}|_V \models \overline{a}$, we get $\overline{w}|_V \in L(\mathbb{T})$, and therefore $\overline{w}|_V \in L(\mathcal{A})$. Let ρ be a run of $\mathcal{A}_{stx}$ on $\overline{c}$. Since $\overline{w}|_V \models \overline{c}$, we have that ρ is a run of $\mathcal{A}$ on $\overline{w}|_V$. From $\overline{w}|_V \in L(\mathcal{A})$ we find that ρ is accepting. Since this is true for all runs ρ and $\mathcal{A}$ has a universal acceptance condition, we finally get $\overline{c} \in L(\mathcal{A}_{stx})$. So, $\overline{a} \in \mathsf{W}_{\mathcal{A},R}$, by definition of $\mathsf{W}_{\mathcal{A},R}$. Finally, we have shown that $L(\mathbb{T}_{stx}) \subseteq \mathsf{W}_{\mathcal{A},R}$.

$\Leftarrow$ Suppose $\mathsf{W}_{\mathcal{A},R}$ is realized by a finite transducer $\mathbb{T}_f = (Q, I, \Sigma_i, \Sigma_o, \delta)$ where $\Sigma_i = Test_{\mathbb{I},R}$, $\Sigma_o = Assign_{\mathbb{I},R} \times Output_{\mathbb{O},R}$ and $\delta : Q \times Test_{\mathbb{I},R} \to Assign_{\mathbb{I},R} \times Output_{\mathbb{O},R} \times Q$. We have $L(\mathbb{T}_f) \subseteq \mathsf{W}_{\mathcal{A},R}$. The finite transducer $\mathbb{T}_f$ can be seen as an R-register transducer $\mathbb{T}$ over $\mathcal{D}$, i.e. such that $\mathbb{T}_{stx} = \mathbb{T}_f$. We show that $\mathbb{T}$ realizes $\mathcal{A}$, i.e. $L(\mathbb{T}) \subseteq L(\mathcal{A})$.

Let $\overline{w} \in L(\mathbb{T})$. The run of $\mathbb{T}$ on $\overline{w}$ induces an action word that we write $\overline{a}$, as well as a word of valuations $\overline{w'} \in (Val_{V\cup R,\mathbb{D}})^\omega$ such that $\overline{w'}|_V = \overline{w}$. We have $\overline{w} \models \overline{a}$ by semantics of register transducer. In particular $\overline{a} \in L(\mathbb{T}_f)$. Remember that we want to show $\overline{w} \in L(\mathcal{A})$. Let ρ be a run of $\mathcal{A}$ on $\overline{w}$. We let $\overline{c}$ be the constraint word induced by ρ, by semantics of constraint word and automata $\overline{w} \models \overline{c}$. Let $\overline{c'} \in \mathcal{MC}_{V\cup R}^{(k)}$ the maximally consistent constraint

word such that $\overline{w'} \models \overline{c'}$ (it is obtained by choosing for every literal and its negation the one that $\overline{w}$ satisfies). As both $\overline{w} \models \overline{a}$ and $\overline{w} \models \overline{c}$, by Lemma 12 we have $\overline{c'} \in \mathsf{join}(\overline{a}, \overline{c})$. Moreover, since $\overline{w'} \models \overline{c'}$, we have $\overline{c'} \in \mathsf{SAT}_{k,V\cup R}$. As $\overline{a} \in L(\mathbb{T}_f)$, so $\overline{a} \in \mathsf{W}_{\mathcal{A},R}$ by hypothesis. By definition of $\mathsf{W}_{\mathcal{A},R}$, we have $\overline{c} \in L(\mathcal{A}_{stx})$. By definition of $\overline{c}$ and since $\mathcal{A}_{stx}$ is universal, ρ is accepting. As this is true for all runs ρ of $\mathcal{A}$ on $\overline{w}$, we get $\overline{w} \in L(\mathcal{A})$. ◀

The next lemma states that $\mathsf{W}_{\mathcal{A},R}$ is ω-regular whenever $\mathcal{D}$ has regular satisfiability and establishes some complexity bounds on the size of the representation of $\mathsf{W}_{\mathcal{A},R}$.

▶ **Lemma 14.** *Let $\mathcal{A}$ be a universal co-Büchi CA with n states. If $\mathcal{D}$ has effective ω-regular satisfiability, then $\mathsf{W}_{\mathcal{A},R}$ is effectively ω-regular: given $k \in \mathbb{N}$ and registers R, if $\mathsf{SAT}_{k,V\cup R}$ is recognizable by a non-deterministic parity automaton with n_s states and c_s colors, then $\mathsf{W}_{\mathcal{A},R}$ is accepted by a universal co-Büchi automaton with $O(n_s \times c_s \times n)$ states.*

Proof sketch. We first consider the following definitions/equalities:

$$\begin{array}{lcl}
\mathsf{A} & = & (Act_{\mathbb{I},\mathbb{O},R})^\omega \times (\mathcal{C}_V^{(k)})^\omega \times (\mathcal{MC}_{V\cup R}^{(k)})^\omega \\
L_{\mathsf{join}} & = & \{(\overline{a},\overline{c},\overline{c'}) \in \mathsf{A} \mid \overline{c'} \in \mathsf{join}(\overline{a},\overline{c})\} \\
\mathsf{W}' & = & \{(\overline{a},\overline{c},\overline{c'}) \in \mathsf{A} \mid (\overline{a},\overline{c},\overline{c'}) \notin L_{\mathsf{join}} \vee \overline{c'} \notin \mathsf{SAT}_{k,V\cup R} \vee \overline{c} \in L(\mathcal{A}_{stx})\} \\
\mathsf{W}_{\mathcal{A},R} & = & \{\overline{a} \in (Act_{\mathbb{I},\mathbb{O},R})^\omega \mid \forall \overline{c} \in (\mathcal{C}_V^{(k)})^\omega, \forall \overline{c'} \in (\mathcal{MC}_{V\cup R}^{(k)})^\omega, (\overline{a},\overline{c},\overline{c'}) \in \mathsf{W}'\}
\end{array}$$

As a consequence, a universal co-Büchi automaton for $\mathsf{W}_{\mathcal{A},R}$ can be derived from one for W'. Let us explain how we build it. First, L_{join} is recognizable by a deterministic (safety) automaton with two states. Then, the *complement* of W' roughly corresponds to the intersection of L_{join} with $\mathsf{SAT}_{k,V\cup R}$ and with the complement of $L(\mathcal{A}_{stx})$. The non-deterministic parity automaton for $\mathsf{SAT}_{k,V\cup R}$ can be translated into a non-deterministic Büchi one in polynomial time. In addition, as $\mathcal{A}$ is a universal co-Büchi CA, the complement of $L(\mathcal{A}_{stx})$ is also recognized by a non-deterministic Büchi automaton. Last, the intersection of two non-deterministic Büchi automata is known to be a non-deterministic Büchi automaton of polynomial size which can built in PTime (see for instance [1]). ◀

Proof sketch of Theorem 10. The Transfer Lemma (Lemma 13) reduces the register-bounded synthesis problem to a synthesis problem over a finite alphabet, whose specification is ω-regular by Lemma 14. This problem is decidable by Büchi-Landweber's Theorem. ◀

Data domains with the completion property. Theorem 10 is established for data domains with effective ω-regular satisfiability. We prove that any data domain satisfying the completion property has ω-regular satisfiability. Intuitively, we build a safety automaton that stores the last constraint read and checks that two consecutive constraints are compatible.

▶ **Lemma 15.** *Let $\mathcal{D}$ be a decidable completable data domain, then $\mathcal{D}$ has ω-regular satisfiability. Moreover, if $\mathcal{D}$ is decidable in ExpTime, then a deterministic parity automaton recognizing $\mathsf{SAT}_{k,X}$ can be constructed that has $exp(k.|X|)$ states and 2 priorities.*

▶ Remark 16. In the statement, we require that the existential first-order theory of $\mathcal{D}$ is decidable. In fact we only need decidability of the satisfiability of conjunctions of constraints and not any formula, this problem is generally easier.

We can now state the main result of this section:

▶ **Theorem 17.** *Let $\mathcal{D}$ be a decidable completable data domain (resp. decidable in ExpTime). Then register-bounded synthesis from $\mathsf{CLTL}(\mathcal{D})$ is decidable (resp. 2ExpTime-c). It is 2ExpTime-hard for any fixed number of registers $r \geq 2$.*

Proof sketch. To prove the upper bound, we first build from a CLTL formula Φ an equivalent CA $\mathcal{A}$, using Proposition 5. Then, using Lemmas 13, 14, 15, realizability of Φ reduces to that of $\mathsf{W}_{\mathcal{A},R}$, which can be recognized by a universal co-Büchi automaton whose size is exponential in $|\Phi|$, k and $|R|$. Last, realizability for specifications expressed by universal co-Büchi automata can be decided in ExpTime [20].

For the lower bound, we reduce the problem of LTL synthesis (over finite alphabets), which is already 2ExpTime-c [21], to bounded register synthesis with two registers, over domain $\mathcal{D}$. Intuitively, we use two distinct data values in order to encode the two boolean values, and add constraints in the formula in order to ensure that the register transducer uses two registers to store these data values all along the execution. ◀

▶ Remark 18. In Theorem 17, the lower bound already holds for two registers, and also if $\mathcal{D}$ has an existential first-order theory decidable in PTime. Moreover, this lower bound holds for *any* data domain, as long as equality is definable.

We have seen in Lemma 1 that $(\mathbb{D}, =, d_0)$ and $(\mathbb{Q}, <, 0)$ are completable data domains. In addition, it is easily seen that the existential first-order theory of $(\mathbb{D}, =, d_0)$ is decidable in NP, as well as for $(\mathbb{Q}, <, 0)$. As a consequence of Theorem 17, we obtain:

▶ **Corollary 19.** *Register-bounded synthesis from* CLTL $(\mathbb{D}, =, d_0)$ *and* CLTL $(\mathbb{Q}, <, 0)$ *is 2ExpTime-complete.*

5 ω-Regularly approximable data domains

Another important setting is the data domain $(\mathbb{N}, <, 0)$. As said before, it is not completable, but worse than that, its set of satisfiable constraint words $\mathsf{SAT}_{k,X}$ is not regular (see e.g. [9,22]). Actually, when considering only finite words, this set is regular, but it turns out that it is not regular when considering infinite words. Here is an example to illustrate this discrepancy. We define $\mathsf{decrease} = (x^{(1)} < x^{(0)} \wedge y^{(1)} = y^{(0)})$ and $\mathsf{reset} = (x^{(1)} = y^{(0)} \wedge y^{(1)} = y^{(0)})$. Then we look at the family of constraint words $\mathsf{reset}.\mathsf{decrease}^{i_1}.\mathsf{reset}.\mathsf{decrease}^{i_2}.\mathsf{reset}.\mathsf{decrease}^{i_3}\ldots$. Depending on the sequence $(i_n)_{n\geq 0}$, the constraint word will or will not be satisfiable: if $(i_n)_{n\geq 0}$ has an upper bound then by picking the first value for y big enough we can build a satisfying valuation, but otherwise the constraint word is not satisfiable. In this section, we will show an extension of the previous framework that allows to capture such data domains.

5.1 General approach

Let X be a set of variables and $k \in \mathbb{N}$. We let $\mathsf{LASSO} \subseteq (\mathcal{MC}_X^{(k)})^\omega$ denote the set of ultimately periodic constraint words (or lasso-shaped word), formally $\mathsf{LASSO} = \{u.v^\omega \mid u, v \in \mathcal{C}_X^{(k)^*}\}$.

▶ **Definition 20.** *We say that $\mathcal{D}$ is* effectively ω-regularly approximable *if there exists a computable function which associates with every k, X, an automaton recognizing an ω-regular language denoted $\mathsf{QSAT}_{k,X} \subseteq (\mathcal{MC}_X^{(k)})^\omega$ such that $\mathsf{SAT}_{k,X} \subseteq \mathsf{QSAT}_{k,X}$ and $\mathsf{SAT}_{k,X} \cap \mathsf{LASSO} = \mathsf{QSAT}_{k,X} \cap \mathsf{LASSO}$.*

▶ Remark 21. $\mathsf{QSAT}_{k,X}$ can be thought of as an over-approximation of $\mathsf{SAT}_{k,X}$ which is exact on lasso-shaped words.

▶ **Theorem 22.** *Let $\mathcal{D}$ be an effectively ω-regularly approximable data domain. Then register-bounded synthesis from specifications expressed as universal co-Büchi CA is decidable.*

Let $\mathcal{A}$ be a universal co-Büchi CA over $\mathcal{D}$, an ω-regularly approximable data domain, and R be a set of registers. Fix $\mathsf{QSAT}_{k,V\cup R} \subseteq (\mathcal{MC}^{(k)}_{V\cup R})^\omega$ as given in Definition 20. We define the following language by:

$$\mathsf{W}^Q_{\mathcal{A},R} = \{\overline{a} \in (Act_{\mathbb{I},\mathbb{O},R})^\omega \mid \forall \overline{c} \in (\mathcal{C}^{(k)}_V)^\omega, (\exists \overline{c'} \in \mathsf{join}(\overline{a},\overline{c}), \overline{c'} \in \mathsf{QSAT}_{k,V\cup R}) \Rightarrow \overline{c} \in L_{stx}(\mathcal{A})\}$$

▶ **Lemma 23.** *$\mathsf{W}_{\mathcal{A},R}$ is realizable (by an FT) iff $\mathsf{W}^Q_{\mathcal{A},R}$ is realizable (by an FT).*

Proof sketch. For the reverse direction, it is easy to verify that the inclusion $\mathsf{SAT}_{k,V\cup R} \subseteq \mathsf{QSAT}_{k,V\cup R}$ entails $\mathsf{W}_{\mathcal{A},R} \supseteq \mathsf{W}^Q_{\mathcal{A},R}$. Thus, if there exists a FT $\mathbb{T}$ that realizes $\mathsf{W}^Q_{\mathcal{A},R}$, *i.e.* $L(\mathbb{T}) \subseteq \mathsf{W}^Q_{\mathcal{A},R}$, then it also realizes $\mathsf{W}_{\mathcal{A},R}$.

To prove the direct implication, we will show that for any FT $\mathbb{T}$, $L(\mathbb{T}) \not\subseteq \mathsf{W}^Q_{\mathcal{A},R}$ entails $L(\mathbb{T}) \not\subseteq \mathsf{W}_{\mathcal{A},R}$. Following the lines of the proof of Lemma 14, and using the same notations, we consider the following definitions/equalities:

$$\begin{array}{lcl} \mathsf{W}'_Q & = & \{(\overline{a},\overline{c},\overline{c'}) \in \mathsf{A} \mid (\overline{a},\overline{c},\overline{c'}) \notin L_{\mathsf{join}} \vee \overline{c'} \notin \mathsf{QSAT}_{k,V\cup R} \vee \overline{c} \in L(\mathcal{A}_{stx})\} \\ \mathsf{W}^Q_{\mathcal{A},R} & = & \{\overline{a} \in (Act_{\mathbb{I},\mathbb{O},R})^\omega \mid \forall \overline{c} \in (\mathcal{C}^{(k)}_V)^\omega, \forall \overline{c'} \in (\mathcal{MC}^{(k)}_{V\cup R})^\omega, (\overline{a},\overline{c},\overline{c'}) \in \mathsf{W}'_Q\} \end{array}$$

In addition, as $\mathsf{QSAT}_{k,V\cup R}$ is ω-regular, we can build a non-deterministic Büchi automaton B accepting the complement $\overline{\mathsf{W}'_Q}$ of W'_Q. Observe that by definition of W' and W'_Q, the equality $\mathsf{SAT}_{k,V\cup R} \cap \mathsf{LASSO} = \mathsf{QSAT}_{k,V\cup R} \cap \mathsf{LASSO}$ entails $\overline{\mathsf{W}'} \cap \mathsf{LASSO} = \overline{\mathsf{W}'_Q} \cap \mathsf{LASSO}$ $(\star)$.

Let $\mathbb{T}$ be an FT such that $L(\mathbb{T}) \not\subseteq \mathsf{W}^Q_{\mathcal{A},R}$: there exist $\overline{a} \in L(\mathbb{T})$, $\overline{c} \in (\mathcal{C}^{(k)}_V)^\omega$, $\overline{c'} \in (\mathcal{MC}^{(k)}_{V\cup R})^\omega$ such that $(\overline{a},\overline{c},\overline{c'}) \notin \mathsf{W}'_Q$, and thus $(\overline{a},\overline{c},\overline{c'}) \in L(B)$. Considering the product $\mathbb{T} \times B$, we can exhibit a lasso shaped word $(\overline{b},\overline{d},\overline{d'}) \in L(B)$ such that $\overline{b} \in L(\mathbb{T})$. Property $(\star)$ entails $(\overline{b},\overline{d},\overline{d'}) \notin \mathsf{W}'$, hence $\overline{b} \notin \mathsf{W}_{\mathcal{A},R}$, and thus $L(\mathbb{T}) \not\subseteq \mathsf{W}_{\mathcal{A},R}$. ◀

▶ Remark 24. Observe that if $\mathsf{W}^Q_{\mathcal{A},R}$ is realizable a FT $\mathbb{T}$, $\mathsf{W}_{\mathcal{A},R}$ is also realizable by $\mathbb{T}$.

Proof sketch of Theorem 22. By Lemmas 13 and 23, we have that $L(\mathcal{A})$ is realizable by a RT iff $\mathsf{W}^Q_{\mathcal{A},R}$ is realizable by a FT. The definition of $\mathsf{W}^Q_{\mathcal{A},R}$ is the same as that of $\mathsf{W}_{\mathcal{A},R}$, while substituting $\mathsf{SAT}_{k,V\cup R}$ with $\mathsf{QSAT}_{k,V\cup R}$. As a consequence, Lemma 14 can be adapted to prove that $\mathsf{W}^Q_{\mathcal{A},R}$ is effectively ω-regular, and we conclude using Büchi-Landweber's Theorem. ◀

5.2 Proving ω-regular approximability

In [16], a similar notion of ω-regular approximability is considered, but the setup is different as they do not start from logic but from register automata. As such, their syntactic input languages are not over constraint words as our paper, but over action words. Their notion is different from ours mainly in that future variables $x^{(k)}$ are restricted to $k = 1$, and that they receive values one at a time. Still, we will show that it is possible to transfer ω-regular approximability results from the setting if [16] to ours.

In the setting of [16], there is a single input, hence $\mathbb{I}$ should be a singleton ($\mathbb{I} = \{\star\}$) and there is no output ($\mathbb{O} = \emptyset$). Last, we let R denote some set of registers. With these choices, we denote by $\mathsf{FEAS}_R \subseteq (Act_{\{\star\},\emptyset,R})^\omega$ the set of action words $\overline{a}$ such that $[\![\overline{a}]\!] \neq \emptyset$. We will also denote by LASSO_{AW} the adaptation of LASSO to the set $(Act_{\{\star\},\emptyset,R})^\omega$.

▶ **Definition 25.** *Let $\mathcal{D}$ be a data domain. We say that $\mathcal{D}$ is effectively AW regularly approximable (AW-RA) if there exists a computable function which associates with every R, an automaton recognizing an ω-regular language denoted $\mathsf{QFEAS}_R \subseteq (Act_{\mathbb{I},\mathbb{O},R})^\omega$, such that $\mathsf{FEAS}_R \subseteq \mathsf{QFEAS}_R$ and $\mathsf{FEAS}_R \cap \mathsf{LASSO}_{AW} = \mathsf{QFEAS}_R \cap \mathsf{LASSO}_{AW}$.*

Now we can state the following result:

▶ **Proposition 26** ([16]). *Each of the following data domain is effectively AW regularly approximable:*

- $(\mathbb{N}, <, 0)$*: natural numbers with linear order*
- $(\mathbb{Z}, <, 0)$*: integers with linear order*
- $(\mathbb{Z}^d, =^d, <^d, 0^d)$*: tuples of integers, with pointwise linear order* ($d \in \mathbb{N}$ *is fixed)*
- $(\Sigma^*, \prec, \epsilon)$*: finite words over* Σ *with the prefix relation*

In addition, for each of these domains, given a set of registers R*, the set* QFEAS_R *is recognized by a non-deterministic parity automaton with* $exp(|R|)$ *states and* $poly(|R|)$ *priorities.*

This follows from different results proven in [16]. First, it is shown in Section 3.2 that for all R, $(\mathbb{N}, <, 0)$ has a witness QFEAS_R of ω-regular approximability recognized by a non-deterministic parity automaton with $exp(|R|)$ states and $poly(|R|)$ priorities. Then, it is shown in Sections 4.2 and 4.3 that the other data domains reduce to $(\mathbb{N}, <, 0)$. In Remark 18, it is explained why these reductions induce a construction for QFEAS_R that preserves its size and number of priorities (only a polynomial blowup occurs).

The next result allows us to transfer these positive results to our setting:

▶ **Lemma 27.** *If a data domain* $\mathcal{D}$ *is effectively AW-RA, then it is effectively* ω*-regularly approximable: if* QFEAS_R *can be recognized by a non-deterministic parity automaton with* $exp(|R|)$ *states and* $poly(|R|)$ *priorities, then* $\mathsf{QSAT}_{k,X}$ *can be recognized by a non-deterministic parity automaton with* $exp((k+1).|X|)$ *states and* $poly((k+1).|X|)$ *priorities.*

Proof sketch. The intuitive idea of the construction is to translate what happens in the setting of constraint sequences into that of action words over inputs of dimension 1. To that end, intuitively, we need to trade the higher dimension of input variables ($|X|$) and the possibility to look at horizon k with longer executions. Each step in the setting of constraint sequences will be simulated by $(k+1).|X|$ steps in the action words setting.

Let $\mathcal{D}$ be an effectively AW regularly approximable data domain. Let $X = \{x_1, \ldots, x_{|X|}\}$ be a set of variables and $k \in \mathbb{N}$. We define the following set of registers:

$$R = \{x_{i,j} \mid 1 \leq i \leq |X|, 0 \leq j \leq k\}$$

In particular, we have $|R| = (k+1).|X|$

Formally, we define a mapping $\Psi : (\mathcal{MC}_X^{(k)})^\omega \to (Act_{\{\star\},\emptyset,R})^\omega$ from constraint sequences over $X^{(k)}$ to action words over R (with the same conditions as above, *i.e.* singleton input variables, and empty set of output variables). We describe how it works on an example. Assume that $X = \{x, y, z\}$ and $k = 1$. We thus have access to six data values, which we will store in six registers. Each step in the constraint sequence setting is simulated by six steps in the action word setting. The way we convert $\overline{w} = c_1 c_2 \ldots$ is depicted on Figure 2.

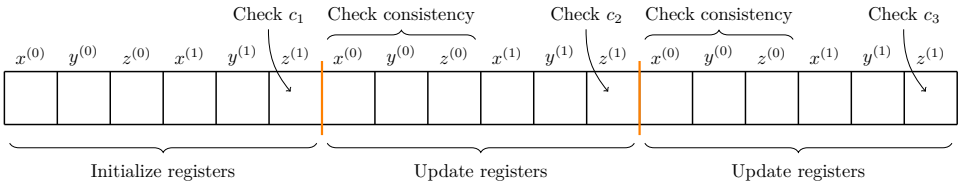


Figure 2 Illustration of the construction of Lemma 27.

During the first six steps, we initialize the registers with the data values that we read. At the end of these steps, we are able to check the first constraint c_1. Then, the data are processed six by six as follows: the first three data should correspond to data seen previously (as $x^{(0)}$ corresponds to $x^{(1)}$ one step before), so we have to check consistency. Still, we update the registers, and at the end of these six steps, we are able to check the constraint c_2.

We claim that this mapping fulfills the two following properties:

(i) $\forall \overline{w} \in (\mathcal{MC}_X^{(k)})^\omega, \overline{w} \in \mathsf{SAT}_{k,X} \Leftrightarrow \Psi(\overline{w}) \in \mathsf{FEAS}_R$

(ii) $\forall \overline{w} \in (\mathcal{MC}_X^{(k)})^\omega, \overline{w} \in \mathsf{LASSO} \Rightarrow \Psi(\overline{w}) \in \mathsf{LASSO}_{AW}$

We let $\mathsf{QSAT}_{k,X} = \Psi^{-1}(\mathsf{QFEAS}_R)$. Property (i) gives $\mathsf{SAT}_{k,X} = \Psi^{-1}(\mathsf{FEAS}_R)$. Thus, $\mathsf{FEAS}_R \subseteq \mathsf{QFEAS}_R$ entails, by monotonicity of the inverse image, $\mathsf{SAT}_{k,X} \subseteq \mathsf{QSAT}_{k,X}$. In addition, Property (ii) easily gives $\mathsf{QSAT}_{k,X} \cap \mathsf{LASSO} \subseteq \mathsf{SAT}_{k,X}$, which implies $\mathsf{SAT}_{k,X} \cap \mathsf{LASSO} = \mathsf{QSAT}_{k,X} \cap \mathsf{LASSO}$ as expected.

Regarding complexity, one can observe that Ψ is realized by an FT $\mathbb{T}_\Psi$ with two states. As $\mathsf{QSAT}_{k,X} = \Psi^{-1}(\mathsf{QFEAS}_R)$, we can build an automaton accepting $\mathsf{QSAT}_{k,X}$ by doing a wreath product between $\mathbb{T}_\Psi$ and an automaton recognizing QFEAS_R, yielding the result, as we have $|R| = (k+1).|X|$. ◀

▶ **Corollary 28.** *For $\mathcal{D} \in \{(\mathbb{N}, <, 0), (\mathbb{Z}, <, 0), (\mathbb{Z}^d, <^d, 0^d), (\Sigma^*, \prec, \epsilon)\}$, register-bounded synthesis from $\mathsf{CLTL}(\mathcal{D})$ is 2ExpTime-c.*

Proof sketch. Let $\mathcal{D}$ be one of these data domains and $\Phi \in \mathsf{CLTL}(\mathcal{D})$. Let $\mathcal{A}$ be a universal co-Büchi CA built from Φ. By Proposition 26 and Lemma 27, a bound on the size of a non-deterministic parity automaton recognizing $\mathsf{QSAT}_{k,V\cup R}$ can be derived. Then, the complexity analysis done in the proof sketch of Theorem 17 can be adapted to show the upper bound. The lower bound follows from the one of Theorem 17 as it does not depend on the data domain (Remark 18). ◀

6 CLTL register-bounded synthesis with partial observation

Partial observation aims to improve the modeling capabilities. While a system may contain numerous variables, the controller usually has access to only a few of them [19]. In this section, we study an extension of CLTL that features partial observation: we split our set of input variables into two subsets, public (visible) inputs $\mathbb{I}_v$ and private (hidden) inputs $\mathbb{I}_h$.

▶ **Example 29.** To illustrate this setting, consider an environment that, at each turn, outputs two public values $\mathsf{in}_1, \mathsf{in}_2$. One of them must be equal to some private (hidden) variable t (target), that the controller aims at identifying infinitely often, using some variable g (guess) that it outputs at each turn. Such a setting can be captured as follows. Let $\mathbb{I}_v = \{\mathsf{in}_1, \mathsf{in}_2\}$, $\mathbb{I}_h = \{\mathsf{t}\}$ and $\mathbb{O} = \{\mathsf{g}\}$ and consider the following formula:

$$\Phi = G\left(\bigvee_{i\in\{1,2\}} \mathsf{in}_i^{(0)} = \mathsf{t}^{(0)}\right) \Rightarrow GF\left(\mathsf{g}^{(0)} = \mathsf{t}^{(0)}\right)$$

As $\mathbb{D}$ is infinite and the way t alternates between in_1 and in_2 is arbitrary, this formula is not realizable. Indeed, for any strategy of the controller (output in_1, resp. in_2), the environment can take the opposite decision (set $\mathsf{t} = \mathsf{in}_2$, resp. $\mathsf{t} = \mathsf{in}_1$). However, if we assume a periodic behavior of the environment, then we obtain the following formula (here with period p):

$$\Phi_{\mathsf{per}} = \left(G\left(\mathsf{t}^{(0)} = \mathsf{t}^{(p)}\right) \wedge G\left(\bigvee_{i\in\{1,2\}} \mathsf{in}_i^{(0)} = \mathsf{t}^{(0)}\right)\right) \Rightarrow GF\left(\mathsf{g}^{(0)} = \mathsf{t}^{(0)}\right)$$

Now, we can show that this formula is realizable by a register transducer with 2 registers, which stores the two first inputs to identify which one repeats after p rounds.

Let $V = \mathbb{I}_v \uplus \mathbb{I}_h \uplus \mathbb{O}$ be a set of variables. We say that a transducer $\mathbb{T}$ with input alphabet $\mathbb{I}_v$ and output alphabet $\mathbb{O}$ *PO-realizes* a specification $L \subseteq (Val_{V,\mathbb{D}})^\omega$ iff $\forall \overline{w_h} \in (Val_{\mathbb{I}_h,\mathbb{D}})^\omega$, $\forall \overline{w} \in L(\mathbb{T})$, $\overline{w} \uplus \overline{w_h} \in L$. This yields the following decision problem:

Register-bounded Partial Observation Synthesis Problem from CLTL($\mathcal{D}$)
Input: A CLTL($\mathcal{D}$) formula Φ over $V = \mathbb{I}_v \uplus \mathbb{I}_h \uplus \mathbb{O}$ and an integer r
Output: A register transducer $\mathbb{T}$ over $(\mathbb{I}_v, \mathbb{O})$ with r registers that PO-realizes Φ, if it exists.

As the specification deals with input variables $\mathbb{I}_v \cup \mathbb{I}_h$, while the transducer only reads inputs in $\mathbb{I}_v$, we need to adapt the transfer lemma. To that end, given an action word $\overline{a}$ over $\mathbb{I}_h$, and an action word $\overline{a'}$ over $\mathbb{I}_v \cup \mathbb{I}_h$, we say that $\overline{a'}$ is a *completion* of $\overline{a}$ if for all $i > 0$, if $a_i = (C_i, \rho_i^{ass}, \rho_i^{out})$ then $a'_i = (C'_i, \rho_i^{ass}, \rho_i^{out})$, with $C'_i \in (\mathcal{MC}^{(k)}_{V \cup R})^\omega$ such that $C_i = C'_i|_{\mathbb{I}_v \cup R}$. We let $\mathsf{compl}_{\mathbb{I}_h}(\overline{a})$ denote the set of completions of $\overline{a}$.

Then, given a universal co-Büchi CA $\mathcal{A}$, we consider the following set:

$$\mathsf{W}^{PO}_{\mathcal{A},R} = \{\ \overline{a} \in (Act_{\mathbb{I}_v,\mathbb{O},R})^\omega \mid \forall \overline{a'} \in \mathsf{compl}_{\mathbb{I}_h}(\overline{a}), \forall \overline{c} \in (\mathcal{C}^{(k)}_V)^\omega, (\exists \overline{c'} \in \mathsf{join}(\overline{a'}, \overline{c}), \overline{c'} \in \mathsf{SAT}_{k,V\cup R}) \Rightarrow \overline{c} \in L_{stx}(\mathcal{A})\ \}$$

We can then adapt the transfer lemma and prove:

▶ **Lemma 30** (Partial observation transfer lemma)**.** *Let $\mathcal{A}$ be a universal co-Büchi CA. Then $L(\mathcal{A})$ is PO-realizable by a RT with $|R|$ registers iff $\mathsf{W}^{PO}_{\mathcal{A},R}$ is realizable by a FT.*

Following the same lines as in Section 4, we prove:

▶ **Theorem 31.** *Let $\mathcal{D}$ be a data domain with effective ω-regular satisfiability. Register-bounded partial observation synthesis from* CLTL($\mathcal{D}$) *is decidable. If, in addition, $\mathcal{D}$ is completable and decidable in ExpTime, then it is 2ExpTime-c.*

▶ **Corollary 32.** *Register-bounded partial observation synthesis from* CLTL $(\mathbb{D}, =, d_0)$ *and* CLTL $(\mathbb{Q}, <, 0)$ *is 2ExpTime-c.*

7 Conclusion

We have shown that when the set of satisfiable constraint sequences over a data domain $\mathcal{D}$ is ω-regular, or ω-regularly approximable, then the register-bounded synthesis problem from CLTL($\mathcal{D}$) is decidable. In addition, we have provided detailed complexity analysis to obtain optimal complexity results for most of the classical data domains studied in the literature. Last, we have also proven that our approach can be generalized to partial observation.

This work opens several perspectives. First, one could investigate natural extensions of this work, for instance by targeting other data domains (*e.g.* sets of natural numbers with inclusion [14]), or other logics over data words (*e.g.* freeze LTL [6]). Another direction consists in trying to lift successful approaches developed for reactive synthesis from the boolean to the data-aware setting. For instance, one could investigate compositional approaches, as well as heuristics based on antichains, as proposed in [13] to develop more efficient symbolic algorithms.

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

2 Ashwin Bhaskar and M. Praveen. Realizability problem for constraint LTL. *Information and Computation*, 2024. arXiv:2207.06708.

3 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 921–962. Springer, 2018. doi:10.1007/978-3-319-10575-8_27.

4 J. Richard Buchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. URL: http://www.jstor.org/stable/1994916.

5 Stéphane Demri. LTL over integer periodicity constraints. *Theor. Comput. Sci.*, 360(1-3):96–123, 2006. doi:10.1016/J.TCS.2006.02.019.

6 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.

7 Stéphane Demri and Karin Quaas. Concrete domains in logics: a survey. *ACM SIGLOG News*, 8(3):6–29, July 2021. doi:10.1145/3477986.3477988.

8 Stéphane Demri and Karin Quaas. Constraint automata on infinite data trees: from CTL(Z)/CTL*(Z) to decision procedures. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPIcs*, pages 29:1–29:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CONCUR.2023.29.

9 Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint LTL. *Information and Computation*, 205(3):380–415, 2007. doi:10.1016/j.ic.2006.09.006.

10 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over linearly ordered data domains. *Formal Methods Syst. Des.*, 61(2):290–337, 2022. doi:10.1007/S10703-023-00435-W.

11 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of data word transducers. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 24:1–24:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.24.

12 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of data word transducers. *Log. Methods Comput. Sci.*, 17(1), 2021. URL: https://lmcs.episciences.org/7279.

13 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods Syst. Des.*, 39(3):261–296, 2011. doi:10.1007/S10703-011-0115-3.

14 Sabína Gulcíková and Ondrej Lengál. Register set automata (technical report). *CoRR*, abs/2205.12114, 2022. doi:10.48550/arXiv.2205.12114.

15 Swen Jacobs, Guillermo A. Pérez, Remco Abraham, Véronique Bruyère, Michaël Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Klara J. Meyer, Thibaud Michaud, Adrien Pommellet, Florian Renkin, Philipp Schlehuber-Caissier, Mouhammad Sakr, Salomon Sickert, Gaëtan Staquet, Clément Tamines, Leander Tentrup, and Adam Walker. The reactive synthesis competition (SYNTCOMP): 2018-2021. *Int. J. Softw. Tools Technol. Transf.*, 26(5):551–567, 2024. doi:10.1007/S10009-024-00754-1.

16 Ayrat Khalimov, Emmanuel Filiot, and Léo Exibard. A generic solution to register-bounded synthesis with an application to discrete orders. *CoRR*, abs/2105.09978, 2021. arXiv:2105.09978.

17 Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 25:1–25:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.25.

18 Ayrat Khalimov, Benedikt Maderbacher, and Roderick Bloem. Bounded synthesis of register transducers. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2018. doi:10.1007/978-3-030-01090-4_29.

19 Orna Kupferman and Moshe Y. Vardi. *Synthesis with Incomplete Information*, pages 109–127. Springer Netherlands, Dordrecht, 2000. doi:10.1007/978-94-015-9586-5_6.

20 Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 531–542. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.66.

21 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. Association for Computing Machinery. doi:10.1145/75277.75293.

22 Luc Segoufin and Szymon Torunczyk. Automata based verification over linearly ordered data domains. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, Dortmund, Germany, March 10-12, 2011*, volume 9 of *LIPIcs*, pages 81–92. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.STACS.2011.81.