Bounded Synthesis of Register Transducers

Ayrat Khalimov[1(✉)], Benedikt Maderbacher[2], and Roderick Bloem[2]

[1] The Hebrew University, Jerusalem, Israel
ayrat.khalimov@gmail.com
[2] Graz University of Technology, Graz, Austria

Abstract. Reactive synthesis aims at automatic construction of systems from their behavioural specifications. The research mostly focuses on synthesis of systems dealing with Boolean signals. But real-life systems are often described using bit-vectors, integers, etc. Bit-blasting would make such systems unreadable, hit synthesis scalability, and is not possible for infinite data-domains. One step closer to real-life systems are register transducers [10]: they can store data-input into registers and later output the content of a register, but they do not directly depend on the data-input, only on its comparison with the registers. Previously [5] it was proven that synthesis of register transducers from register automata is undecidable, but there the authors considered transducers equipped with the unbounded queue of registers. First, we prove the problem becomes decidable if bound the number of registers in transducers, by reducing the problem to standard synthesis of Boolean systems. Second, we show how to use quantified temporal logic, instead of automata, for specifications.

1 Introduction

Reactive synthesis [2] frees hardware and software developers from tedious and error-prune coding work. Instead, the developer specifies the desired behaviour of a system, and a synthesizer produces the actual code. The research in reactive synthesis is mostly focused on synthesis of transducers dealing with *Boolean* inputs and outputs. However, most programs and hardware designs use not only Booleans, but also bit-vectors, integers, reals. Bit-blasting into Booleans makes synthesized programs unreadable and hinders the synthesis scalability.

One step closer to real-life systems are register transducers [10]. Such transducers are equipped with registers; they can read the data-input from an infinite domain; they can store the data-input into a register and later output it; they do not depend on the exact data-input value, but on its comparison with the registers. Thus, a transition of a register transducer can say "in state q: if the data-input not equals to register #1, then output the value of register #1, store the data-input into register #2, and go into state q'". Examples of a register transducer and automaton are in Figs. 2 and 1.

In [5], the authors introduced the problem of synthesis of register transducers. But their transducers are equipped with an *unbounded queue* of registers: they

S. K. Lahiri and C. Wang (Eds.): ATVA 2018, LNCS 11138, pp. 494–510, 2018.
https://doi.org/10.1007/978-3-030-01090-4_29

can push the data-input into the queue, and later compare the data-input with the values in the queue. For specifications, the authors use register automata with a fixed number of registers (thus, no queue). The authors show that the synthesis problem is undecidable; the proof relies on unboundedness of the queue.

We prove the problem becomes decidable if bound the number of registers in transducers. Namely, we reduce synthesis of k-register transducers wrt. register automata to synthesis of Boolean transducers wrt. Boolean automata, i.e., to standard synthesis. The reduction relies on two ideas.

The first (folklore) idea is: instead of tracking the exact register values and data-inputs, track only the *equivalences* between register values and the data-input. The second idea is: instead of checking automaton non-emptiness, we check automaton non-emptiness *modulo words of k-register transducers*.

In the second part, we suggest a temporal logic that "works well" with our approach. Among several logics suitable to the context of infinite data [3,4,9,14], we have chosen IPTL [14] (called VLTL in [9]), because of its naturalness. Using this logic, we can state properties like $\forall \mathit{d} \in \mathcal{D} : \mathsf{G}(i = \mathit{d} \rightarrow \mathsf{F}(o = \mathit{d}))$: "every data-value appearing on the input eventually appears on the output". We show how to convert a formula in this logic into a register automaton (in incomplete way; there can be no complete way) that can be used by our synthesis approach.

2 Definitions

Fix a *data-domain* $\mathcal{D}$ throughout the paper, which is an infinite set of elements (*data-values*). Calligraphic writing like i, o, d, z denotes data-variables or objects closely related to them. Sets of such objects are also written in calligraphic, like $\mathcal{D}$, $\mathcal{R}$, $\mathcal{P}$, etc. Define $\mathbb{N} = \{1, 2, ...\}$, $\mathbb{N}_0 = \{0, 1, 2, ...\}$, $[k] = \{1, ..., k\}$ for $k \in \mathbb{N}$; $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$, and we often use the subscripted variants, $\mathbb{B}_i = \mathbb{B}_o = \mathbb{B}$, to clarify when $\mathbb{B}$ is related to object i or o. For an automaton A, let $L(A)$ denote the set of its accepting words.

2.1 Register Automata

A register automaton works on words from $(2^P \times \mathcal{D}^{\mathcal{P}})^\omega$, where P is a set of Boolean signals and $\mathcal{P}$ is a set of data-signals. To simplify the presentation, we assume there are only two data-signals ($\mathcal{P} = \{i, o\}$), which makes the words to be from $(2^P \times \mathcal{D}^2)^\omega$. When reading a word, a register automaton can store the value of data-signal i into its registers. Later it can compare the content of its registers with the current value of i. Register automata do not depend on actual data-values—only on the comparison with the register values. Below is a formal definition.

A *(universal co-Büchi/non-deterministic Büchi) word automaton with k registers* is a tuple $A = \langle P, \mathcal{P}, \mathcal{R}, \mathit{d}_0, Q, q_0, \delta, F \rangle$, where

- P is a set of *Boolean signals*;
- $\mathcal{P} = \{i, o\}$ is a set of *data-signals*;

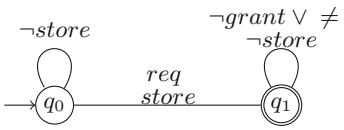

Fig. 1. A universal co-Büchi 1-register automaton: $P = \{req, grant\}$, $\mathcal{R} = \{\imath\}$, $F = \{q_1\}$. The labels $\neg store$ and $store$ have a special meaning: $store$ means that the automaton stores the value of data-input i into register $\imath$; $\neg store$ means it does not. The expression $o \neq \imath$ means that the component $\mathbb{B}_o$ of the transition is *false*. For guards and Boolean signals, the labeling is symbolic. Formally, the set of transitions is $\{(q_0, p, b_i, b_o, false, q_0) : (b_i, b_o) \in \mathbb{B}^2, p \in 2^P\} \cup \{(q_0, p, b_i, b_o, true, q_1) : (b_i, b_o) \in \mathbb{B}^2, req \in p \in 2^P\} \cup \{(q_1, p, b_i, b_o, false, q_1) : (b_i, b_o) \in \mathbb{B}^2, p \in 2^P, grant \notin p \vee b_o = false\}$.

- $\mathcal{R} = \{\imath_1, ..., \imath_k\}$ is a set of *registers*;
- $\mathit{d}_0 \in \mathcal{D}$ is an *initial data-value* for every register;
- Q is the set of *states* and $q_0 \in Q$ is an *initial state*;
- $F \subseteq Q$ is a set of *accepting states*;
- $\delta : Q \times 2^P \times \mathbb{B}_i^k \times \mathbb{B}_o^k \to 2^{\mathbb{B}^k \times Q}$ is a *transition function*. Intuitively, in a state, an automaton reads a finite letter from 2^P (which describes all Boolean signals whose current value is true) and a data-letter from D^2 (a data-value for i and a data-value for o). Then the automaton compares the data-letter with the content of the registers. Depending on this comparison (component $\mathbb{B}_i^k \times \mathbb{B}_o^k$, called *guard*), the automaton transits into several (for universal automaton) or one of (for non-deterministic automaton) successor states, and for each successor state, stores the value of data-signal i into one, several, or none of the registers (defined by component $\mathbb{B}^k$, called *assignment* or *store*).

An example of a register automaton is in Fig. 1.

A *configuration* is a tuple $(q, \bar{\mathit{d}}) \in Q \times \mathcal{D}^k$, and (q_0, d_0^k) is *initial*. A *path* is an infinite sequence $(q_0, \bar{\mathit{d}}_0) \xrightarrow{(l_0, i_0, o_0, \bar{a}_0)} (q_1, \bar{\mathit{d}}_1) \xrightarrow{(l_1, i_1, o_1, \bar{a}_1)} ...$ such that for every $j \in \mathbb{N}_0$:

- $q_j \in Q$, $\bar{\mathit{d}}_j \in \mathcal{D}^k$, $l_j \in 2^P$, $i_j \in \mathcal{D}$, $o_j \in \mathcal{D}$, and $\bar{a}_j \in \mathbb{B}^k$;
- $(q_{j+1}, \bar{a}_j) \in \delta\big(q_j, l_j, i_j = \bar{\mathit{d}}_j[1], ..., i_j = \bar{\mathit{d}}_j[k], o_j = \bar{\mathit{d}}_j[1], ..., o_j = \bar{\mathit{d}}_j[k]\big)$;
- $\bar{\mathit{d}}_0 = \mathit{d}_0^k$; and
- for every $n \in [k]$: $\bar{\mathit{d}}_{j+1}[n] = \begin{cases} i_j & \text{if } \bar{a}_j[n] = true, \\ \bar{\mathit{d}}_j[n] & \text{otherwise.} \end{cases}$

Let $\Sigma = 2^P \times \mathcal{D}^2$. A *word* is a sequence from Σ^ω. A word is *accepted* by a universal co-Büchi automaton iff every path —whose projection into Σ equals to the word— does not visit a state from F infinitely often; otherwise the word is *rejected*. A word is *accepted* by a non-deterministic Büchi automaton iff there is a path —whose projection into Σ equals to the word— that visits a state from F infinitely often; otherwise the word is *rejected*. For example, the universal co-Büchi 1-register automaton in Fig. 1 accepts the word $(\{req\}, 5_i, *_o)(\{req, grant\}, 6_i, 5_o)(\{grant\}, *_i, 6_o)(\emptyset, *_i, *_o)^\omega$, where $\mathcal{D} = \mathbb{N}_0$, we

write subscripts i and o for clarity, and $*$ is anything from $\mathcal{D}$ (not necessary the same). The automaton describes the words where every *req* is followed by *grant* with the data-value of o being equal to the data-value of i at the moment of the request. Such words can be described by a formula $\forall \mathit{d} \in \mathcal{D} : \mathsf{G}\big(\mathit{req} \wedge i = \mathit{d} \rightarrow \mathsf{XF}(\mathit{grant} \wedge o = \mathit{d})\big)$, but we postpone the discussion of logic until Sect. 4.

2.2 Register Transducers

Register transducers is an extension of standard transducers (Mealy machines) to an infinite domain. A register transducer can store the input data-value into its registers. It can only output the data-value that is currently stored in one of its registers. Similarly to register automata, the transitions of register transducers depend on the comparison of the data-input with the registers, but not on the actual data-values. Let us define register transducers formally.

A *k-register transducer* is a tuple $T = \langle I, O, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathit{d}_0, S, s_0, \tau \rangle$ where:

- I and O are sets of Boolean signals, called *Boolean inputs* and *outputs*;
- $\mathcal{I}$ and $\mathcal{O}$ are sets of data-signals, called *data-inputs* and *data-outputs*; we assume that $\mathcal{I} = \{i\}$ and $\mathcal{O} = \{o\}$.
- S is a (finite or infinite) set of *states* and $s_0 \in S$ is *initial*;
- $\mathcal{R} = \{r_1, ..., r_k\}$ is a set of *registers*;
- $\mathit{d}_0 \in \mathcal{D}$ is an *initial data-value* for every register;
- $\tau : S \times 2^I \times \mathbb{B}_i^k \rightarrow (2^O \times [k] \times \mathbb{B}^k \times S)$ is a *transition function*. Intuitively, from a state the transducer reads the values of the Boolean inputs (component 2^I) and compares the content of the registers with the data-value of i (component $\mathbb{B}_i^k$, called *guard*). Depending on that information, the transducer transits into a unique successor state (component S), stores the data-value of i into one, several, or none of the registers (component $\mathbb{B}^k$, called *assignment* or *store*), outputs a value for each Boolean output (component 2^O), and outputs a data-value stored in one of the registers (component $[k]$).

Figure 2 shows an example of a register transducer.

A *configuration* is a tuple $(s, \bar{\mathit{d}}) \in Q \times \mathcal{D}^k$; (s_0, d_0^k) is called *initial*. A *path* is a sequence $(s_0, \bar{\mathit{d}}_0) \xrightarrow{(i_0, o_0, i_0, o_0, \bar{a}_0)} (s_1, \bar{\mathit{d}}_1) \xrightarrow{(i_1, o_1, i_1, o_1, \bar{a}_1)} \ldots$ where for every $j \in \mathbb{N}_0$:

- $s_j \in S$, $\bar{\mathit{d}}_j \in \mathcal{D}^k$, $i_j \in 2^I$, $o_j \in 2^O$, $i_j \in \mathcal{D}$, $o_j \in \mathcal{D}$, $\bar{a}_j \in \mathbb{B}^k$;
- let $(\mathit{out}, \mathit{out}, \mathit{store}, \mathit{succ}) = \tau(s_j, i_j, i_j = \bar{\mathit{d}}_j[1], ..., i_j = \bar{\mathit{d}}_j[k])$. Then:
- $s_{j+1} = \mathit{succ}$;
- $\bar{a}_j = \mathit{store}$;
- $o_j = \bar{\mathit{d}}_j[\mathit{out}]$;
- $o_j = \mathit{out}$;
- $\bar{\mathit{d}}_0 = \mathit{d}_0^k$; and
- for every $n \in [k]$: $\bar{\mathit{d}}_{j+1}[n] = \begin{cases} i_j & \text{if } \bar{a}_j[n] = \mathit{true}, \\ \bar{\mathit{d}}_j[n] & \text{otherwise.} \end{cases}$

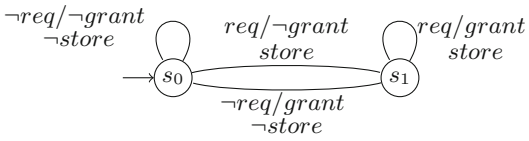

Fig. 2. A 1-register transducer: $I = \{req\}, O = \{grant\}$, $\mathcal{R} = \{\iota\}$. The meaning of *store* and $\neg store$ is as in the previous figure. The labeling wrt. guards and Boolean signals is symbolic. The transducer always outputs the value of its only register (not shown). Formally, the set of transitions is $\{(s_0, \emptyset, b_i, \emptyset, 1, false, s_0) : b_i \in \mathbb{B}\} \cup \{(s_0, \{req\}, b_i, \emptyset, 1, true, s_1) : b_i \in \mathbb{B}\} \cup \{(s_1, \{req\}, b_i, \{grant\}, 1, true, s_1) : b_i \in \mathbb{B}\} \cup \{(s_1, \emptyset, b_i, \{grant\}, 1, false, s_0) : b_i \in \mathbb{B}\}$.

Notice that a value of the data-output refers to the current register values, not the updated ones. I.e., outputting a data-value happens before storing.

For example, a path of the register transducer in Fig. 2 can start with $(s_0, 0) \xrightarrow{(\{req\}, \emptyset, 5_i, 0_o, true)} (s_1, 5) \xrightarrow{(\{req\}, \{grant\}, 6_i, 5_o, true)} (s_1, 6) \xrightarrow{(\emptyset, \{grant\}, 4_i, 6_o, false)} (s_0, 6)$, where we assumed that $\mathcal{D} = \mathbb{N}_0$, $\mathit{0}_0 = 0$, and the subscripts i and o are for clarity.

A *word* is a projection of a transducer path into $2^{I \cup O} \times \mathcal{D}^2$. A register transducer *satisfies* a register automaton A, written $T \models A$, iff all transducer words are accepted by the automaton. For example, the register transducer from Fig. 2 satisfies the automaton from Fig. 1.

2.3 Synthesis Problem

Model checking and cutoffs The *model-checking problem* is:

- Given: a register transducer T, a universal co-Büchi register automaton A.
- Return: "yes" if $T \models A$, otherwise "no".

The model-checking problem is decidable, which follows from the following. Kaminski and Francez [10, Prop.4] proved the following *cutoff result* (adapted to our notions): if a data-word over an infinite domain $\mathcal{D}$ is accepted by a non-deterministic Büchi k-register automaton, then there is an accepting data-word over a finite domain $\mathcal{D}_{k+1}$ of size $k+1$. (Actually, their result is for words of finite length, but can be extended to infinite words.) Further, if we look at a given universal co-Büchi k_A-register automaton A as being non-deterministic Büchi $\widetilde{A}$, then $L(\widetilde{A}) = \overline{L(A)}$, i.e., it describes the error words. To do model checking, as usual, (1) build the product of the $\widetilde{A}$ and a given k_T-register transducer T, then (2) check its emptiness and return "the transducer is correct" iff the product is empty. The product is easy to build, this is an easy extension of the standard product construction, we note only that it is a non-deterministic Büchi $(k_A + k_T)$-register automaton. Finally, to check emptiness of the product we can use the cutoff result, namely, restrict the data-domain to have $(k_A + k_T + 1)$ data-values. This reduces product emptiness to standard emptiness of register-less automata.

The case of deterministic Rabin register automata and transducers with more than single data-input and data-output was studied in [12], but the proof idea is similar.

In this paper we focus on the synthesis problem defined below.

Synthesis. The *bounded synthesis problem* is:

- Given: a register-transducer interface (the number of registers k_T, Boolean and data-inputs, Boolean and data-outputs), a universal co-Büchi register automaton A.
- Return: a k_T-register transducer T of a given interface such that $T \models A$, otherwise "unrealizable".

If the number of registers k_T is not given (thus we ask to find any such k_T which makes the problem realizable, or return "unrealizable" if no such k_T exists), then we get the (finite but unbounded) *synthesis problem*.

A related synthesis problem (let us call it "infinite synthesis problem") was studied in [5], but for a slightly different model of register transducers. There, the transducers operate an unbounded queue of registers (thus, it may use an infinite number of registers). They prove the infinite synthesis problem is undecidable and suggest an incomplete synthesis approach.

In the next sections, we show that the bounded synthesis problem is decidable, and suggest an approach that reduces it to the synthesis problem of register-less transducers wrt. register-less automata. The (unbounded) synthesis problem is left open.

But before proceeding to our solution, let us remark why the cutoff result does not immediately give a complete synthesis procedure.

Remark 1 (Cutoffs and synthesis). The cutoff result makes the data-domain finite, so let the values of the registers be part of the transducer states. Then a transducer has to satisfy the three conditions below, where condition (3) explains why the cutoff does not work with this naive approach.

(1) "The register values are updated according to transducer store actions."
Introduce new Boolean outputs describing the current values of the transducer registers, and new Boolean outputs describing the store action. Then it is easy to encode the above requirement using a register-less automaton.

(2) "The value of the data-output always equals the value of one of the registers."
With the Boolean outputs introduced in item (1), this can be easily encoded using a register-less automaton.

(3) "The transitions depend on the guard, but not on the value of data-input."
When considered alone, this requirement can be implemented using the partial-information synthesis approach [11], where we search for a transducer that can access the guard, but not the actual value of data-input. But the partial-information synthesis approach does not allow for having partial information for transitions (needed to implement item (3)), yet full information for outputs (needed to implement items (1) and (2)).

Nevertheless, with the cutoff it is easy to get an *incomplete* synthesis approach with SMT-based bounded synthesis [6] that allows you to fine-tune transition and output functions dependencies.

3 Solving the Bounded Synthesis Problem

Recall that given a non-deterministic Büchi register automaton A, we search for a k_T-register transducer T such that $T \models \neg A$ (equiv., $T \models \tilde{A}$ where $\tilde{A}$ is dual to A and thus it is universal co-Büchi). Our approach is 5 points long.

(**1**) We start by defining a Boolean associate $A_{\mathbb{B}}$ of a non-deterministic Büchi register automaton A, which is a standard register-less non-deterministic Büchi automaton derived from the description of A. Of course, we cannot directly use the Boolean associate $A_{\mathbb{B}}$ to answer questions about A, because $A_{\mathbb{B}}$ lacks the semantics of A. We also define a Boolean associate $T_{\mathbb{B}}$ for every register transducer T. In the end, we will synthesize $T_{\mathbb{B}}$ that satisfies a certain register-less automaton. For examples of such associates, look at the automaton and transducer on Figs. 1 and 2 as being standard, register-less. (**2**) We introduce a verifier automaton V, which tracks the equivalences between the registers $\mathcal{R}^A$ of A: two registers fall into the same equivalence class iff they hold the same data-value. The automaton $A_{\mathbb{B}}@V$ is $A_{\mathbb{B}}$ enhanced with this equivalence-class information. It has enough information to answer the questions like non-emptiness A and model checking wrt. A. This is because every Boolean path of $A_{\mathbb{B}}@V$ corresponds to some data-path in A, and vice versa (which was not the case for $A_{\mathbb{B}}$ and A). But $A_{\mathbb{B}}@V$, or rather the dual universal automaton $\overline{A_{\mathbb{B}}@V}$, is not suited for synthesis —we cannot synthesize from $\overline{A_{\mathbb{B}}@V}$— roughly because the transducer should not control the store actions of the underlying A, while the automaton $\overline{A_{\mathbb{B}}@V}$ gives a transducer such a control. (**3**) We add k_T fresh registers $\mathcal{R}^T$ to A that will be controlled by a transducer. To this end, we define the automaton $A \otimes T^{all}$. Additionally, the automaton T^{all} filters out data-words that do not belong to any of the transducers (e.g., data-words that have a value for o that was not seen before on i). (**4**) We enhance the Boolean associate $(A \otimes T^{all})_{\mathbb{B}}$ of $A \otimes T^{all}$ with information about equivalences between the registers $\mathcal{R}^T$ *and* $\mathcal{R}^A$; the resulting automaton is called $(A \otimes T^{all})_{\mathbb{B}}@W$. (**5**) Finally, we hide the information that should not be visible to a transducer, namely information related to the automaton registers $\mathcal{R}^A$. The resulting automaton is called $H = hide_A((A \otimes T^{all})_{\mathbb{B}}@W)$ and it is such that $\exists T : T \models \tilde{A}$ iff $\exists T_{\mathbb{B}} : T_{\mathbb{B}} \models \tilde{H}$, where $\tilde{H}$ is dual to H.

3.1 Boolean Associates of Register Automata and Transducers

The transition functions of k-register automata do not contain any infinite objects—data-values appear only in the semantics. Let us define Boolean associates of register automata and transducers.

Given a k-register automaton $A = \langle P, \mathcal{P}, \mathcal{R}, \mathcal{A}_0, Q, q_0, \delta, F \rangle$, let *Boolean automaton* $A_{\mathbb{B}} = \langle P_{\mathbb{B}}, Q, q_0, \delta_{\mathbb{B}}, F \rangle$ be a standard register-less automaton where:

- let $G_i = \{g_{i\imath_1}, ..., g_{i\imath_k}\}$, $G_o = \{g_{o\imath_1}, ..., g_{o\imath_k}\}$, $Asgn = \{a_{\imath_1}, ..., a_{\imath_k}\}$. Then:
- $P_{\mathbb{B}} = P \cup G_i \cup G_o \cup Asgn$,
- $\delta_{\mathbb{B}} : Q \times 2^{P_{\mathbb{B}}} \to 2^Q$ contains $(q, l \cup g_i \cup g_o \cup a, q') \in \delta_{\mathbb{B}}$ iff $(q, l, \bar{b}_i, \bar{b}_o, \bar{a}, q') \in \delta$, where $l \in 2^P$, $g_i \in 2^{G_i}$, $g_o \in 2^{G_o}$, $a \in 2^{Asgn}$, $\bar{b}_i = (g_{i\imath_1} \in g_i, ..., g_{i\imath_k} \in g_i) \in \mathbb{B}^k$, $\bar{b}_o = (g_{o\imath_1} \in g_o, ..., g_{o\imath_k} \in g_o) \in \mathbb{B}^k$, $\bar{a} = (a_{\imath_1} \in a, ..., a_{\imath_k} \in a) \in \mathbb{B}^k$. Informally, we take the assignment component (on the right side) of δ and move it to the left side of $\delta_{\mathbb{B}}$, and introduce new Boolean signals to describe the Boolean components.

For convenience, we say that a letter $g_i \in 2^{G_i}$ *encodes* the guard $(g_{i\imath_1} \in g_i, ..., g_{i\imath_k} \in g_i) \in \mathbb{B}^k$, and vice versa; similarly for a letter from 2^{G_o} and 2^{Asgn}.

A *Boolean path* is an infinite sequence $q_0 \xrightarrow{l_0 \cup g_{i0} \cup g_{o0} \cup a_0} q_1 \xrightarrow{l_1 \cup g_{i1} \cup g_{o1} \cup a_1} ...$ from $(Q \times 2^{P_{\mathbb{B}}})^\omega$ that satisfies $\delta_{\mathbb{B}}$. When necessary to distinguish paths of register automata (which are in $(Q \times \mathcal{D}^k \times 2^P \times \mathcal{D}^2)^\omega$) from Boolean paths, we call the former *data-paths*. A data-path $(q_0, \bar{\ell}_0) \xrightarrow{(l_0, i_0, o_0, \bar{a}_0)} (q_1, \bar{\ell}_1) \xrightarrow{(l_1, i_1, o_1, \bar{a}_1)} ...$ *corresponds* to a Boolean path $q_0 \xrightarrow{l_0 \cup g_{i0} \cup g_{o0} \cup a_0} q_1 \xrightarrow{l_1 \cup g_{i1} \cup g_{o1} \cup a_1} ...$ where g_{ij} encodes the guard $(i_j = \bar{\ell}_j[1], ..., i_j = \bar{\ell}_j[k])$, g_{oj} encodes the guard $(o_j = \bar{\ell}_j[1], ..., o_j = \bar{\ell}_j[k])$, and $a_j \in 2^{Asgn}$ encodes $\bar{a}_j \in \mathbb{B}^k$, for $j \in \mathbb{N}_0$. From the definition of paths of register automata on page 3, it follows that for every path of a register automaton, there exists a path in the associated Boolean automaton to which the data-path corresponds. Consider the reverse direction, where we say that a Boolean path *corresponds* to a data-path iff the data-path corresponds to it. The reverse direction does not necessarily hold: there is a register automaton A (e.g., with 2 registers) where some Boolean paths of $A_{\mathbb{B}}$ do not have a corresponding data-path in A. This is because the letters of a Boolean path can describe contradictory guards. For example, let a transition in a Boolean path have $\bar{a} = (true, true)$, meaning that in a data-path the value of data-input is stored into the registers $\imath_1$ and $\imath_2$. Hence, in the next transition of the data-path, $i = \imath_1 \Leftrightarrow i = \imath_2$ must hold, but the Boolean path may have $g_i = \{g_{i\imath_2}\}$ (describing the guard $i \neq \imath_1 \wedge i = \imath_2$). Thus, we got the following.

Observation 1.

- *For every register automaton A, every data-path in A has exactly one corresponding Boolean path in $A_{\mathbb{B}}$.*
- *There exists a register automaton A where some Boolean paths of $A_{\mathbb{B}}$ do not correspond to any data-path of A.*

A *Boolean word* is a projection of a Boolean path into $2^{P_{\mathbb{B}}}$; note that it contains information about assignment actions.

Similarly we define Boolean transducers. Given a k-register transducer $T = \langle I, O, \mathcal{G}, \mathcal{O}, \mathcal{R}, \ell_0, S, s_0, \tau \rangle$, a *Boolean transducer* $T_{\mathbb{B}} = \langle I_{\mathbb{B}}, O_{\mathbb{B}}, S, s_0, \tau_{\mathbb{B}} \rangle$ is a standard register-less transducer where: $I_{\mathbb{B}} = I \cup G_i$, $G_i = \{g_{i\imath_1}, ..., g_{i\imath_k}\}$, $O_{\mathbb{B}} = O \cup Asgn \cup O_k$, $Asgn = \{a_{\imath_1}, ..., a_{\imath_k}\}$, and O_k has enough Boolean signals to encode the numbers $[k]$. The transition function $\tau_{\mathbb{B}} : S \times 2^{I_{\mathbb{B}}} \to S \times 2^{O_{\mathbb{B}}}$ contains $(s, l \cup g_i, o \cup o_k \cup a, s')$ iff $(s, l, \bar{b}_i, o, \tilde{o}_k, \bar{a}, s') \in \tau$ where $s, s' \in S$, $l \in 2^I$, $a \in 2^{Asgn}$

encodes $\bar{a} \in \mathbb{B}^k$, $g_i \in 2^{G_i}$ encodes $\bar{b}_i \in \mathbb{B}^k$, and $o_k \in 2^{O_k}$ encodes $\tilde{o}_k \in [k]$. A *Boolean path* is an infinite sequence $s_0 \overset{l_0 \cup g_{i0}, o_0 \cup o_{k0} \cup a_0}{\longrightarrow} s_1 \overset{l_1 \cup g_{i1}, o_1 \cup o_{k1} \cup a_1}{\longrightarrow} \ldots$ from $(S \times 2^{I_{\mathbb{B}}} \times 2^{O_{\mathbb{B}}})^\omega$ that satisfies $\tau_{\mathbb{B}}$.

Because every register transducer can be viewed as a register automaton, a similar observation holds for the register transducers.

3.2 Verifier to Remove Inconsistent Guards (V_k and $A_{\mathbb{B}}@V_k$)

We introduce the automaton called verifier that filters out the Boolean paths of $A_{\mathbb{B}}$ that do not correspond to any data-paths.

$\mathbf{V_k}$. Given $k \in \mathbb{N}$, the *verifier* is a deterministic looping register-less automaton $V_k = \langle P_V, \Pi, \pi_0, \delta_V \rangle$ where

- Π is the set of all possible partitions of $\{\imath_1, ..., \imath_k\}$; the initial state $\pi_0 = \{\{\imath_1, ..., \imath_k\}\}$ contains the only partition. Later, we will a partition-state to track if the registers have the same value.
- $P_V = G_i \cup G_o \cup Asgn$ where $G_i = \{g_{i\imath_1}, ..., g_{i\imath_k}\}$, $G_o = \{g_{o\imath_1}, ..., g_{o\imath_k}\}$, $Asgn = \{a_{\imath_1}, ..., a_{\imath_k}\}$.
- $\delta_V : \Pi \times 2^{P_V} \to \Pi$ contains $\pi \overset{g_i \cup g_o \cup a}{\longrightarrow} \pi'$ where:
  - the guard-letter $g_i \cup g_o$ respects the current partition:
    - for every $\imath_m = \imath_n$ of π (i.e., belonging to the same partition):
      $g_{i\imath_m} \in g_i \Leftrightarrow g_{i\imath_n} \in g_i$ and $g_{o\imath_m} \in g_o \Leftrightarrow g_{o\imath_n} \in g_o$;
    - for every $\imath_m \neq \imath_n$ of π (i.e., belonging to different partitions):
      $g_{i\imath_m} \in g_i \Rightarrow g_{i\imath_n} \notin g_i$ and $g_{o\imath_m} \in g_o \Rightarrow g_{o\imath_n} \notin g_o$;
  - the successor partition respects the assignment-letter a, formalized as follows. For every m, n in $[k]$, let e_{mn} denote that π contains $\imath_m = \imath_n$, and e'_{mn} is for π'. The value e'_{mn} is uniquely defined:

    $$e'_{mn} = (a_{\imath_m} \wedge a_{\imath_n}) \vee (\neg a_{\imath_m} \wedge a_{\imath_n} \wedge g_{i\imath_m}) \vee (a_{\imath_m} \wedge \neg a_{\imath_n} \wedge g_{i\imath_n}) \vee (\neg a_{\imath_m} \wedge \neg a_{\imath_n} \wedge e_{mn}).$$

    This definition, together with the previous item, ensures that all e'_{mn} together form a partition (e.g., it is impossible to get $e'_{1,2} \wedge e'_{2,3} \wedge \neg e'_{1,3}$).
- The acceptance condition (not shown in the tuple) defines every path (infinite by definition) to be accepting; hence, every word that has a path in the automaton is accepted.

An example of a verifier is in Fig. 3.

$\mathbf{A_{\mathbb{B}}@V_k}$. Given a verifier $V_k = \langle P^V, Q^V, q_0^V, \delta^V \rangle$ and a register-less non-deterministic Büchi automaton $A_{\mathbb{B}} = \langle P^A, Q^A, q_0^A, \delta^A, F^A \rangle$, let $A_{\mathbb{B}}@V$ denote the non-deterministic Büchi automaton $\langle P, Q, q_0, \delta, F \rangle$ where:

- $P = P^V \cup P^A$;
- $Q = Q^V \times Q^A$, $q_0 = (q_0^V, q_0^A)$;
- $\delta : Q \times 2^P \to 2^Q$ has $\big((q_V, q_A), p, (q'_V, q'_A)\big)$ iff $(q_V, p \cap 2^{P^V}, q'_V) \in \delta^V$ and $(q_A, p \cap 2^{P^A}, q'_A) \in \delta^A$; and
- $F = Q^V \times F^A$.

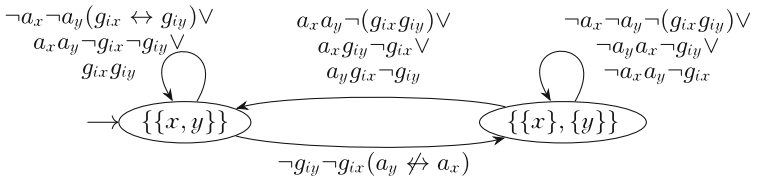

Fig. 3. A verifier automaton (a register-less looping automaton) for 2-register automata with $\mathcal{R} = \{x, y\}$. The edges have symbolic labels. Later, the left state $\{\{x, y\}\}$ will be used to denote that the registers x and y store the same value, while the right state $\{\{x\}, \{y\}\}$ will denote that they store different values. The automaton has similar restrictions for o (not shown).

Hence, $L(A_{\mathbb{B}}@V_k) = L(V_k) \cap L(A_{\mathbb{B}})$. Since $P^A = P' \cup G_i \cup G_o \cup Asgn$ (where P' are the Boolean signals of the register automaton A) and $P^V = G_i \cup G_o \cup Asgn$, the automaton $A_{\mathbb{B}}@V_k$ works on words from $(P' \cup G_i \cup G_o \cup Asgn)^\omega$. The words of $A_{\mathbb{B}}@V_k$ that do not fall out of V_k are called *consistent*, otherwise *inconsistent*.

Observation 2. *For every non-deterministic Büchi k-register automaton A:*

- *Every data-path of A has exactly one corresponding Boolean path in $A_{\mathbb{B}}@V_k$.*
- *Every Boolean path of $A_{\mathbb{B}}@V_k$ has either one or infinitely many corresponding data-paths in A.*

Corollary 1. *For every non-deterministic Büchi k-register automaton A: $A_{\mathbb{B}}@V_k$ has an accepted Boolean word $\Leftrightarrow$ A has an accepted data-word.*

This result, namely, decidability of non-emptiness non-deterministic Büchi register automata, was earlier established in [10, Thm.1] using cutoffs (we discussed cutoffs on page 5). Our verifier uses a similar insight, but allows us to easily extend it to the context of synthesis.

3.3 Focusing on Transducer Data-Words (T^{all} and $A \otimes T^{all}$)

For the next step to become clear, we need to look ahead at Theorem 1. There, we will be interested in data-words that belong to some register transducer, rather than general data-words. Recall that the data-words of register transducers require the signal o to have the value that appeared before in the signal i. The automaton T^{all} introduced below ensures this.

$\mathbf{T^{all}}$. Given $k_T \in \mathbb{N}$, T^{all} is a non-deterministic looping k_T-register automaton $\langle P, \mathcal{P}, \mathcal{R}, \ell_0, Q, q_0, \delta, F \rangle$ with $P = I \cup O$, $\mathcal{P} = \{i, o\}$, $Q = F = \{q_0\}$. The transition function $\{q_0\} \times 2^P \times \mathbb{B}_i^{k_T} \times \mathbb{B}_o^{k_T} \to \{q_0\} \times 2^{\mathbb{B}^{k_T}}$, for every $\bar{g}_i \in \mathbb{B}_i^{k_T}$ and $\bar{g}_o \in \{\bar{g} \in \mathbb{B}^{k_T} \| \exists j. \bar{g}[j] = true\}$, contains $(q_0, \bar{a})$ for every $\bar{a} \in \mathbb{B}^{k_T}$. I.e., it ensures that the value of data-output o comes from a register and it does not restrict the assignment action.

Observation 3. *Fix $k_T \in \mathbb{N}$. $\forall w \in (2^{I \cup O} \times \mathcal{D}^2)^\omega$: $w \models T^{all} \Leftrightarrow \exists T$: $w \models T$, where T is a k_T-register transducer (possibly with $|S| = \infty$).*

In the observation, T might need infinitely many states, because an accepting path of T^{all} on w might exhibit "irregular" storing behaviour, which cannot be expressed by a finite-state transducer (recall transducers are deterministic). This is a minor technical detail though.

$A \otimes T^{all}$. The product $A \otimes T^{all}$ of a non-deterministic Büchi k_A-register automaton $A = \langle P, \mathscr{P}, \mathcal{R}^A, \mathit{\ell}_0, Q^A, q_0^A, \delta^A, F^A \rangle$ and $T^{all} = \langle P, \mathscr{P}, \mathcal{R}^T, \mathit{\ell}_0, Q^T, q_0^T, \delta^T, F^T \rangle$ with k_T registers is a non-deterministic Büchi $(k_A + k_T)$-register automaton $\langle P, \mathscr{P}, \mathcal{R}, \mathit{\ell}_0, Q, q_0, \delta, F \rangle$ where $Q = Q^A \times Q^T$, $q_0 = (q_0^A, q_0^T)$, $F = F^A \times F^T$, $\mathcal{R} = \mathcal{R}^A \dot{\cup} \mathcal{R}^T$, and the transition function $\delta : Q \times 2^P \times \mathbb{B}_i^{k_A+k_T} \times \mathbb{B}_o^{k_A+k_T} \to 2^{Q \times \mathbb{B}^{k_A+k_T}}$ respects both the transitions functions, δ^A and δ^T.

Observation 4. *$A \otimes T^{all}$ has an accepting word $\Leftrightarrow$ A has an accepting word that belongs to some k_T-register transducer (possibly with $|S| = \infty$).*

3.4 Synthesis-tailored Verifier ($AT_{\mathbb{B}}@W$)

For brevity, let AT denote $A \otimes T^{all}$, and let $AT_{\mathbb{B}}$ be its Boolean associate.

The automaton $AT_{\mathbb{B}}@W$ introduced in this section closely resembles $AT_{\mathbb{B}}@V_k$ and $A_{\mathbb{B}}@V_k$, but it is better suited for synthesis.

Recall from Sect. 3.1 that every $T_{\mathbb{B}}$ generates words from $(2^{I \cup G_i^T} \times 2^{O \cup Asgn^T \cup O_{k_T}})^\omega$, where $Asgn^T = \{a_{\imath_1^T}, ..., a_{\imath_{k_T}^T}\}$, $G_i^T = \{g_{i\imath_1^T}, ..., g_{i\imath_{k_T}^T}\}$, and O_{k_T} has enough Boolean signals to encode the numbers $[k_T]$. In synthesis, we want our target specification automaton to have the same alphabet. The automaton $AT_{\mathbb{B}}@V_k$ uses o-guards instead of O_k signals, hence we introduce the automaton $AT_{\mathbb{B}}@W$ (we do not introduce W separately).

Suppose we have $AT_{\mathbb{B}}@V_k = \langle P, Q, q_0, \delta, F \rangle$ with $P = I \cup O \cup G_i^T \cup G_i^A \cup G_o^T \cup G_o^A \cup Asgn^T \cup Asgn^A$ and $\delta : Q \times 2^P \to 2^Q$. The automaton $AT_{\mathbb{B}}@W = \langle P', Q, q_0, \delta', F \rangle$ has the same states, but $P' = (P \setminus G_o^T) \cup O_k$ and the transition function δ' is derived from δ as follows. For every $(\pi, q) \xrightarrow{(i,o,g_i,g_o,a)} (\pi', q')$ of δ (where π and π' are partitions of $\mathcal{R}^A \cup \mathcal{R}^T$, q and q' are states of $AT_{\mathbb{B}}$, $i \in 2^I$ and $o \in 2^O$, $g_i \in 2^{G_i}$ and $g_o \in 2^{G_o}$, $a \in 2^{Asgn}$):

- let $J = \{j_1, ..., j_l\} \subset \mathbb{N}$ be such that g_o contains $o = \imath_j^T$ for every $j \in J$;
- for every $j \in J$, add to δ' the transition $(\pi, q) \xrightarrow{(i,o,g_i,\tilde{j},a)} (\pi', q')$, where $\tilde{j} \in 2^{O_{k_T}}$ encodes the number j.
- Note that if J is empty (g_o requires that $\bigwedge_{t \in [k_T]} o \neq \imath_t^T$), then we do not add transitions to δ', because no transducer can produce such a value for o.

Observation 5. *$AT_{\mathbb{B}}@W$ has an accepting Boolean word $\Leftrightarrow$ A has an accepting data-word that belongs to some k_T-register transducer (possibly with $|S| = \infty$).*

3.5 Synthesis Using Automaton $hide_A(AT_{\mathbb{B}}@W)$

We cannot use $AT_{\mathbb{B}}@W$ for synthesis, because it uses Boolean signals that are not visible to transducers (underlined): $I \cup O \times \underline{G_i^A} \cup G_i^T \cup \underline{G_o^A} \cup O_{k_T} \cup \underline{Asgn^A} \cup Asgn^T$. Let us show that the simple hiding operation resolves the issue.

Given $AT_{\mathbb{B}}@W = \langle P, Q, q_0, \delta, F\rangle$ with $P = I \cup O \cup G_i^A \cup G_i^T \cup G_o^A \cup O_{k_T} \cup Asgn^A \cup Asgn^T$, the automaton $hide_A(AT_{\mathbb{B}}@W)$ is a non-deterministic Büchi automaton $\langle P', Q, q_0, \delta', F\rangle$ with $P' = I \cup O \cup G_i^T \cup O_{k_T} \cup Asgn^T$, and the transition function $\delta' : Q \times 2^I \times 2^O \times 2^{G_i^T} \times 2^{O_{k_T}} \times 2^{Asgn^T} \to 2^Q$ is such that in every transition $q \xrightarrow{(i,o,g_i^T,j,a^T)} Q'$ the destination set $Q' \subseteq Q$ contains all successor states of every transition of $AT_{\mathbb{B}}@W$ starting in q and having the same common labels:

$$Q' = \bigcup_{g_i^A \in 2^{G_i^A}, g_o^A \in 2^{G_o^A}, a^A \in 2^{Asgn^A}} \delta(q, i, o, g_i^A, g_i^T, g_o^A, j, a^T, a^A).$$

Observation 6. *For every non-deterministic Büchi register automaton A and $k_T \in \mathbb{N}$:*

- *Every path of $AT_{\mathbb{B}}@W$ corresponds to exactly one path of $hide_A(AT_{\mathbb{B}}@W)$.*
- *Every path of $hide_A(AT_{\mathbb{B}}@W)$ corresponds to at least one path of $AT_{\mathbb{B}}@W$.*

Observations 5 and 6 result in the following.

Lemma 1. *For every register transducer T and non-deterministic Büchi register automaton A: $\big(\exists w \in L(T) : w \models A\big) \Leftrightarrow \big(\exists w_{\mathbb{B}} \in L(T_{\mathbb{B}}) : w_{\mathbb{B}} \models hide_A(AT_{\mathbb{B}}@W)\big)$.*

Theorem 1. *For every universal co-Büchi register automaton $\tilde{A}$ and $k_T \in \mathbb{N}$:*

$$\big(\exists T : T \models \tilde{A}\big) \;\Leftrightarrow\; \big(\exists T_{\mathbb{B}} : T_{\mathbb{B}} \models \neg hide_A(AT_{\mathbb{B}}@W)\big),$$

where T is a k_T-register transducer, A is dual to $\tilde{A}$ (thus it is non-deterministic Büchi), and $\neg hide_A(AT_{\mathbb{B}}@W)$ is an automaton expressing the complemented language of $hide_A(AT_{\mathbb{B}}@W)$ (e.g., it can be the dual universal co-Büchi automaton).

The right side of the theorem (the standard Boolean synthesis problem) holds iff it holds for finite-state transducers (e.g., see [13]). Hence we get:

Corollary 2. *A given instance of the bounded synthesis problem is realizable $\Leftrightarrow$ it is realizable by a finite-state ($|S| < \infty$) register transducer.*

Finally, Fig. 4 depicts the relation between the languages of utilized automata. It illustrates that the approach makes use of determinizable superset of AT (see point 4 in the figure). (In the resulting automaton $hide_A(AT_{\mathbb{B}}@W)$, when we treat it as a register automaton, the store actions are controlled by transducers *and* are deterministic, i.e., we can associate with each store action a Boolean letter controlled by a transducer.)

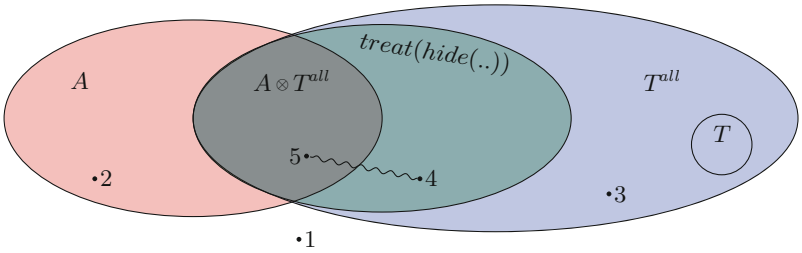

Fig. 4. Inclusion between languages. The language of $treat(hide(..))$ denotes the language of $hide_A(AT_{\mathbb{B}}@W)$ treated as a register automaton. The existence of points 1, 2, 3, and 5 is trivial. Figure 5 justifies the existence of point 4. The snake line indicates "if a transducer T has point 4, then it also has point 5"(follows from Lemma 1 and observations). If $T \models \neg A$ for some k_T-register transducer, then its language is located as shown by T.

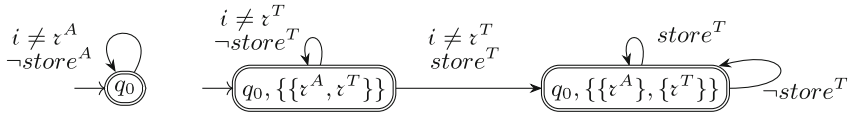

Fig. 5. The automata to show the existence of point 4 in Fig.4. On the left is a non-deterministic Büchi 1-register automaton A: it accepts the words where the signal i is never equal to $\mathit{0}_0$ (and no restrictions on the values of o). On the right is $hide_A(AT_{\mathbb{B}}@W)$ where $k_T = 1$: if treated as a register automaton, it accepts the words whose first value of i is not $\mathit{0}_0$ (plus some restrictions on o). Hence, $L(treat(hide_A(AT_{\mathbb{B}}@W))) \not\subset L(A \otimes T^{all})$. The labels related to o are omitted.

4 Using Temporal Logic in our Synthesis Approach

We proceed to the topic of synthesis of register transducers from a temporal logic. Section 4.1 defines the first-order linear temporal logic with equality, LTL(EQ)[1] and its variants ∃LTL(EQ) and ∀LTL(EQ), known as IPTL in [14] and VLTL in [9]. Then Sect. 4.2 defines register-guessing automata that can express ∃LTL(EQ) formulas.

The sound and complete conversion of ∃LTL(EQ) into register-guessing automata is described in Sect. 4.3. Then Sect. 4.4 describes a sound but incomplete conversion of register-guessing automata into register automata, which implies the sound but incomplete conversion of ∃LTL(EQ) into register automata. The latter automata are consumed by our synthesizer.

Unless explicitly stated, all automata are non-deterministic Büchi.

[1] The name LTL(EQ) is inspired by the names of logics in SMT-LIB [1].

4.1 LTL(EQ) (Also known as IPTL [14] and VLTL [9])

Let $\mathcal{X}$ be a set of data-variables and P be a set of Boolean propositions. An *LTL(EQ) (prenex-quantified) formula* Φ is of the form (for every $k \in \mathbb{N}$):

$$\begin{aligned} \Phi &= \forall x_1 ... x_k.cond.\varphi | \exists x_1 ... x_k.cond.\varphi \\ cond &= true | x \neq x | cond \wedge cond \\ \varphi &= true | p | i = x | o = x | \neg\varphi | \varphi \wedge \varphi | \varphi \,\mathsf{U}\, \varphi | \mathsf{X}\varphi \end{aligned}$$

where $x_1, ..., x_k, x \in \mathcal{X}$, $p \in P$, i and o are two data-propositions, and all the data-variables appearing in φ are quantified. As usual, define $\mathsf{G}\varphi$ to be $\neg\mathsf{F}\neg\varphi$, $\mathsf{F}\varphi = true\,\mathsf{U}\,\varphi$, $\varphi_1 \vee \varphi_2$ is $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2$ is $\neg\varphi_1 \vee \varphi_2$, and *false* is $\neg true$.

Given $w = w_1 w_2 ... \in (2^P \times \mathcal{D}^{\{i,o\}})^\omega$, define the satisfaction $w \models \Phi$:

- $w \models \forall x_1 ... x_k.cond.\varphi$ iff for all $d_1, ..., d_k \in \mathcal{D}$ either $cond[x_1 \leftarrow d_1, ..., x_k \leftarrow d_k]$ does not hold or $w \models \varphi[x_1 \leftarrow d_1, ..., x_k \leftarrow d_k]$;
- $w \models \exists x_1 ... x_k.cond.\varphi$ iff there exists $d_1, ..., d_k \in \mathcal{D}$ such that $cond[x_1 \leftarrow d_1, ..., x_k \leftarrow d_k]$ holds and $w \models \varphi[x_1 \leftarrow d_1, ..., x_k \leftarrow d_k]$;
- let ϕ have the same grammar as φ except that instead of data-variables it has data-values; then
- $w \models true$;
- $w \not\models \phi$ iff $\neg(w \models \phi)$;
- $w \models \neg\phi$ iff $\neg(w \models \phi)$;
- $w \models p$ iff $p \in w_1$;
- $w \models \phi_1 \wedge \phi_2$ iff $w \models \phi_1$ and $w \models \phi_2$;
- for every $d \in \mathcal{D}$, $w \models i = d$ iff in w_1 the data-proposition i has the value d; similarly for o;
- for $i \in \mathbb{N}$, let $w_{[i:]}$ denote w's suffix $w_i w_{i+1} ...$; then
- $w \models \mathsf{X}\phi$ iff $w_{[2:]} \models \phi$; and
- $w \models \phi_1 \,\mathsf{U}\, \phi_2$ iff $\exists i \in \mathbb{N} : \big((w_{[i:]} \models \phi_2) \wedge (\forall j < i : w_{[j:]} \models \phi_1)\big)$.

Let $\exists$LTL(EQ) denote LTL(EQ) where formulas have existential quantifiers only, and use $\forall$LTL(EQ) for universally quantified LTL(EQ) formulas.

4.2 Register Automata with Guessing but Without Storing

In this section we define a variation of register automata that have a non-deterministically chosen initial register values that cannot be rewritten afterwards. Such automata are a restricted version of variable automata [8].

A *k-register-guessing automaton* is a tuple $A = \langle P, \mathcal{P}, \mathcal{R}, Q, q_0, \delta, F, E \rangle$ (notice: no initial register value d_0 and a new element E) with transition function δ of the form $Q \times 2^P \times \mathbb{B}_i^k \times \mathbb{B}_o^k \rightarrow 2^Q$ (notice: no assignment component on the right), where $E \subseteq \mathcal{R} \times \mathcal{R}$ is an *inequality set*[2], while all other components are like for register automata. A path is defined similarly to a path of a register automaton, except that

[2] We can get away without using E (by encoding it into δ), but it proved to be convenient in Sect. 4.4.

– an initial configuration $(q_0, \bar{d}_0) \in \{q_0\} \times \mathcal{D}^k$ of the path is arbitrary provided that $\bar{d}_0$ satisfies the inequality set: $\forall(\tau_i, \tau_j) \in E : \bar{d}_0[i] \neq \bar{d}_0[j]$; and
– the automaton never stores to the registers.

An accepting word is defined as for register automata.

4.3 Converting ∃LTL(EQ) into Register-Guessing Automata

This section describes the sound and complete conversion of ∃LTL(EQ) formulas into register-guessing automata. The fact that a conversion is possible was noted in [7, Sec.4], however they did not describe the conversion itself.

Consider an ∃LTL(EQ) formula $\Phi = \exists x_1 ... x_k . cond . \varphi(i, o, x_1, ..., x_k)$. We will use the notions of $w_{\mathbb{B}}$ and $\varphi_{\mathbb{B}}$ defined below.

($w_{\mathbb{B}}$) Given a word $w \in (2^P \times \mathcal{D}^2)^\omega$ and $x_1, ..., x_k \in \mathcal{D}$, let $w_{\mathbb{B}} \in (2^P \times \mathbb{B}_i^k \times \mathbb{B}_o^k)^\omega$ be the word derived from w by replacing every value of i and o in w by the vectors of values, $(i = x_1, ..., i = x_k)$ and $(o = x_1, ..., o = x_k)$.
($\varphi_{\mathbb{B}}$) In $\varphi(i, o, x_1, ..., x_k)$, replace every expression $i = x_i$ with $g_{i\tau_i}$ and every expression $o = x_i$ with $g_{o\tau_i}$. This introduces $2k$ new Boolean propositions, let $P_{\mathbb{B}} = P \cup \{g_{i\tau_1}, ..., g_{i\tau_k}\} \cup \{g_{o\tau_1}, ..., g_{o\tau_k}\}$. Let $\varphi_{\mathbb{B}}(g_{i\tau_1}, ..., g_{i\tau_k}, g_{o\tau_1}, ..., g_{o\tau_k})$ be the resulting LTL formula over Boolean propositions $P_{\mathbb{B}}$.

To convert a formula $\exists x_1 ... x_k . cond . \varphi$ into a k-register-guessing automaton A do the following (**conversion-1**).

– Convert $\varphi_{\mathbb{B}}$ into an NBW automaton $A_{\mathbb{B}} = \langle P_{\mathbb{B}}, Q, q_0, \delta_{\mathbb{B}}, F \rangle$ using standard approaches. Thus, for every $w_{\mathbb{B}} \in 2^{P_{\mathbb{B}}}$: $w_{\mathbb{B}} \models A_{\mathbb{B}}$ iff $w_{\mathbb{B}} \models \varphi_{\mathbb{B}}$.
– Treat $A_{\mathbb{B}}$ as a k-register-guessing automaton $A = \langle P, \mathscr{P}, \mathscr{R}, Q, q_0, \delta, F, E \rangle$, where E is derived from *cond*.

Observation 7. *For every* $w \in (2^P \times \mathcal{D}^2)^\omega$*:* $w \models A \;\Leftrightarrow\; w \models \exists x_1 ... x_k . cond . \varphi$.

4.4 Converting ∃LTL(EQ) into Register Automata

In this section, we describe a sound but incomplete conversion of register-guessing automata into standard register automata. Together with conversion-1 from the previous section, this gives the conversion of ∃LTL(EQ) formulas into register automata. Note that no complete conversion of ∃LTL(EQ) formulas into register automata exists: for example, the formula $\exists x . \mathsf{G}(i \neq x)$ has no equivalent register automaton, although there is an equivalent register-guessing automaton.

In automata, we will use the definition of δ that is symbolic instead of explicit, hence the transition functions of k-register-guessing automata and of k-register automata are of the form $Q \times 2^P \times G \to 2^Q$ and $Q \times 2^P \times G \to 2^{Q \times \mathbb{B}^k}$, (previously we had $\mathbb{B}_i^k \times \mathbb{B}_o^k$ instead of G), where $g \in G$ has the form $g = true \| g \wedge g \| i \sim \tau \| o \sim \tau$ where $\sim$ denotes $=$ or $\neq$, and $\tau \in \mathscr{R}$. Using the symbolic definition rather than the explicit one is crucial in making our conversion more applicable

Given a k-register-guessing automaton $A = \langle P, \mathscr{P}, \mathscr{R}, Q, q_0, \delta, F, E \rangle$, construct the k-register automaton $A' = \langle P, \mathscr{P}, \mathscr{R}, \bar{d}_0, Q', q_0', \delta', F' \rangle$ (**conversion-2**):

- $Q' = Q \times \mathbb{B}^k$. The Boolean component encodes, for every $\imath_i \in \mathcal{R}$, whether the register $\imath_i$ is assigned a value or not (ignoring the initial values). The initial state $q'_0 = (q_0, \mathit{false}, ..., \mathit{false})$. We call a register $\imath_i$ with $b_i = \mathit{false}$ *uninitialized*.
- $F' = \{(q, b_1, ..., b_k) \in Q' \| q \in F\}$.
- For every state $(q, b_1, ..., b_k) \in Q'$ and A-transition $q \xrightarrow{(l,g)} q'$ ($l \in 2^P$, $g \in G$):
  - If $g = \mathit{true}$, then add to δ' the transition $(q, b_1, ..., b_k) \xrightarrow{(l,g,\mathit{false}^k)} (q', b_1, ..., b_k)$.
  - Otherwise, do the following.
    * Abort point: if there exists $i \in [k]$ such that $b_i = \mathit{false}$ and g contains $i \neq \imath_i$ or $o \sim \imath_i$, then abort. Because the register $\imath_i$ is uninitalized ($b_i = \mathit{false}$), we cannot know the valuation of $i \neq \imath_i$ or $o \neq \imath_i$. In contrast, if the guard g contains $i = \imath_i$, we can assume that it holds and store i into $\imath_i$ (we cannot do this for $o = \imath_i$, because the automata do not allow for storing o).
    * Add to δ' the transition $(q, b_1, ..., b_k) \xrightarrow{(l,g',a)} (q', b'_1, ..., b'_k)$ where for every $i \in [k]$:
      · $b'_i = \mathit{true}$ iff $b_i = \mathit{true}$ or g contains $i = \imath_i$.
      · The action a stores i into $\imath_i$ iff g contains $i = \imath_i$ and $b_i = \mathit{false}$.
      · The guard g' contains $i \sim \imath_i$ iff g contains $i \sim \imath_i$ and $b_i = \mathit{true}$; similarly for $o \sim \imath_i$.
    * Finally, we account for the inequality set E and update g' as follows. For every $(\imath_i, \imath_j) \in E$: if $b_i = \mathit{true}$ and the action a contains $\imath_j = i$, then add to g' the expression $i \neq r_i$.
    (Here we assume that the A-transition is not contradictory, namely, it is not the case that $\exists (\imath_i, \imath_j) \in E : b_i = \mathit{false} \wedge b_j = \mathit{false} \wedge (i = \imath_i) \in g \wedge (i = \imath_j) \in g$. Such transitions cannot be executed in A and can be removed beforehand.)
- Note that the automaton A' never compares i nor o with a register that was uninitialized. Therefore, the component ℓ_0 of A' can be anything from $\mathcal{D}$.

The automaton A' has $|Q'| = |Q| \cdot 2^k$, but the number of reachable states is $|Q| \cdot k$.

Observation 8. *If conversion-2 succeeds, then $L(A) = L(A')$.*

Theorem 2. *Given an $\exists LTL(EQ)$ $\Phi = \exists x_1, ..., x_k.cond.\varphi$. If conversion-1 and conversion-2 succeed, then $L(\Phi) = L(A')$.*

References

1. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB) (2016). www.SMT-LIB.org
2. Church, A.: Logic, arithmetic, and automata. In: International Congress of Mathematicians (Stockholm, 1962), pp. 23–35. Institute Mittag-Leffler, Djursholm (1963)
3. Demri, S., D'Souza, D., Gascon, R.: Temporal logics of repeating values. J. Log. Comput. **22**(5), 1059–1096 (2012). https://doi.org/10.1093/logcom/exr013

4. Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. ACM Trans. Comput. Log. **10**(3), 16:1–16:30 (2009). https://doi.org/10.1145/1507244.1507246
5. Ehlers, R., Seshia, S.A., Kress-Gazit, H.: Synthesis with identifiers. In: McMillan, K.L., Rival, X. (eds.) VMCAI 2014. LNCS, vol. 8318, pp. 415–433. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54013-4_23
6. Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT **15**(5–6), 519–539 (2013)
7. Frenkel, H., Grumberg, O., Sheinvald, S.: An automata-theoretic approach to modeling systems and specifications over infinite data. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NFM 2017. LNCS, vol. 10227, pp. 1–18. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57288-8_1
8. Grumberg, O., Kupferman, O., Sheinvald, S.: Variable automata over infinite alphabets. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 561–572. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13089-2_47
9. Grumberg, O., Kupferman, O., Sheinvald, S.: Model checking systems and specifications with parameterized atomic propositions. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, pp. 122–136. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33386-6_11
10. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**(2), 329–363 (1994). http://www.sciencedirect.com/science/article/pii/0304397594902429
11. Kupferman, O., Vardi, M.: Synthesis with incomplete informatio. In: 2nd International Conference on Temporal Logic, pp. 91–106. Manchester (1997)
12. Lazić, R., Nowak, D.: A unifying approach to data-independence. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 581–596. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44618-4_41
13. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, pp. 179–190. ACM Press, Austin, Texas, USA, January 11–13 (1989). https://doi.org/10.1145/75277.75293
14. Wolper, P.: Expressing interesting properties of programs in propositional temporal logic. In: Proceedings of the 13th POPL, pp. 184–193. ACM, New York, NY, USA (1986). https://doi.org/10.1145/512644.512661