Bounded Synthesis of Register Transducers

Ayrat Khalimov[1], Benedikt Maderbacher[2], Roderick Bloem[2]

[2] Graz University of Technology, Austria
[1] Hebrew University, Israel

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 [12]: 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 [6] 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 [3] 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 [12]. 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 Figures 2 and 1.

In [6], the authors introduced the problem of synthesis of register transducers. But their transducers are equipped with an *unbounded queue* of registers: they 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*. Every such word can be enhanced with assignment actions of the transducer that resulted in producing the word.

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 [17,11,5,4], we have chosen IPTL [17] (called VLTL in [11]), because of its naturalness. Using this logic, we can state properties like $\forall \mathcal{d} \in \mathcal{D} : \mathsf{G}(\mathcal{i} = \mathcal{d} \to \mathsf{F}(\mathcal{o} = \mathcal{d}))$: "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 $\mathcal{i}$, $\mathcal{o}$, $\mathcal{d}$, $\mathcal{r}$ 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, $\mathbb{B}_{\mathcal{i}} = \mathbb{B}_{\mathcal{o}} = \mathbb{B}$, to clarify when $\mathbb{B}$ is related to object $\mathcal{i}$ or $\mathcal{o}$. 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 ($\mathcal{P} = \{\mathcal{i}, \mathcal{o}\}$), 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 $\mathcal{i}$ into its registers. Later it can compare the content of its registers with the current value of $\mathcal{i}$. 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 $A = \langle P, \mathcal{P}, \mathcal{R}, \mathcal{d}_0, Q, q_0, \delta, F \rangle$, where

- P is a set of *Boolean signals*;
- $\mathcal{P} = \{\mathcal{i}, \mathcal{o}\}$ is a set of *data-signals*;
- $\mathcal{R} = \{\mathcal{r}_1, ..., \mathcal{r}_k\}$ is a set of *registers*;
- $\mathcal{d}_0 \in \mathcal{D}$ 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*;

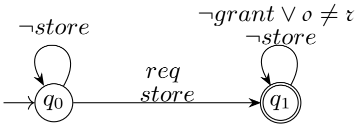

Fig. 1: A universal co-Büchi 1-register automaton: $P = \{req, grant\}$, $\mathcal{R} = \{\mathscr{r}\}$, $F = \{q_1\}$. The labels $\neg store$ and $store$ have a special meaning: $store$ means that the automaton stores the value of data-input i into register $\mathscr{r}$; $\neg store$ means it does not. The expression $o \neq \mathscr{r}$ 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 $\big\{(q_0, p, b_i, b_o, \mathit{false}, q_0) : (b_i, b_o) \in \mathbb{B}^2, p \in 2^P\big\} \cup \big\{(q_0, p, b_i, b_o, \mathit{true}, q_1) : (b_i, b_o) \in \mathbb{B}^2, req \in p \in 2^P\big\} \cup \big\{(q_1, p, b_i, b_o, \mathit{false}, q_1) : (b_i, b_o) \in \mathbb{B}^2, p \in 2^P, grant \notin p \vee b_o = \mathit{false}\big\}$.

- $\delta : Q \times 2^P \times \mathbb{B}_i^k \times \mathbb{B}_o^k \to 2^{\mathbb{B}^k \times Q}$ 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 Figure 1.

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

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

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 register 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 register 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 register automaton in Figure 1 accepts the word $(\{req\}, 5_i, *_o)(\{req, grant\}, 6_i, 5_o)(\{grant\}, *_i, 6_o)(\varnothing, *_i, *_o)^\omega$, 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

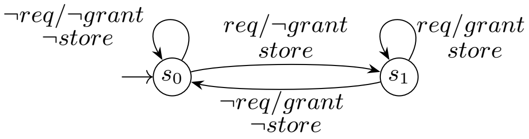

Fig. 2: A 1-register transducer: $I = \{req\}, O = \{grant\}$, $\mathscr{R} = \{\mathscr{r}\}$. The meaning of *store* and $\neg$*store* is as in the previous figure. The labeling wrt. guards and Boolean signals is symbolic. The transducer always outputs the value of its only register (not shown). Formally,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

request. Such words can be described by a formula $\forall \mathscr{d} \in \mathcal{D} : \mathsf{G}\big(req \land \mathscr{i} = \mathscr{d} \to \mathsf{X}\,\mathsf{F}(grant \land \mathscr{o} = \mathscr{d})\big)$, but we postpone the discussion of logic until Section 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 $T = \langle I, O, \mathscr{I}, \mathscr{O}, \mathscr{R}, \mathscr{d}_0, S, s_0, \tau \rangle$ where:

- I and O are sets of Boolean signals, called *Boolean inputs* and *outputs*;
- $\mathscr{I}$ and $\mathscr{O}$ are sets of data-signals, called *data-inputs* and *data-outputs*; we assume that $\mathscr{I} = \{\mathscr{i}\}$ and $\mathscr{O} = \{\mathscr{o}\}$.
- S is a (finite or infinite) set of *states* and $s_0 \in S$ is *initial*;
- $\mathscr{R} = \{\mathscr{r}_1, ..., \mathscr{r}_k\}$ is a set of *registers*;
- $\mathscr{d}_0 \in \mathcal{D}$ is an *initial data-value* for every register;
- $\tau : S \times 2^I \times \mathbb{B}^k_{\mathscr{i}} \to (2^O \times [k] \times \mathbb{B}^k \times S)$ 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 $\mathscr{i}$ (component $\mathbb{B}^k_{\mathscr{i}}$, called *guard*). Depending on that information, the transducer transits into a unique successor state (component S), stores the data-value of $\mathscr{i}$ 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 $(s, \bar{\mathscr{d}}) \in Q \times \mathcal{D}^k$; $(s_0, \mathscr{d}_0^k)$ is called *initial.* A *path* is a sequence $(s_0, \bar{\mathscr{d}}_0) \xrightarrow{(i_0, o_0, \mathscr{i}_0, \mathscr{o}_0, \bar{a}_0)} (s_1, \bar{\mathscr{d}}_1) \xrightarrow{(i_1, o_1, \mathscr{i}_1, \mathscr{o}_1, \bar{a}_1)}$ ... where for every $j \in \mathbb{N}_0$:

- $s_j \in S$, $\bar{\mathscr{d}}_j \in \mathcal{D}^k$, $i_j \in 2^I$, $o_j \in 2^O$, $\mathscr{i}_j \in \mathcal{D}$, $\mathscr{o}_j \in \mathcal{D}$, $\bar{a}_j \in \mathbb{B}^k$;
- let $(out, \mathscr{out}, store, succ) = \tau(s_j, i_j, \mathscr{i}_j = \bar{\mathscr{d}}_j[1], ..., \mathscr{i}_j = \bar{\mathscr{d}}_j[k])$. Then:
- $s_{j+1} = succ$;

- $\bar{a}_j = store$;
- $o_j = \bar{d}_j[out]$;
- $\underline{o}_j = out$;
- $\bar{d}_0 = d_0^k$; and
- for every $n \in [k]$: $\bar{d}_{j+1}[n] = \begin{cases} i_j & \textit{if } \bar{a}_j[n] = true, \\ \bar{d}_j[n] & \textit{otherwise.} \end{cases}$

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 Figure 2 can start with $(s_0, 0) \xrightarrow{(\{req\},\varnothing,5_i,0_o,true)} (s_1, 5) \xrightarrow{(\{req\},\{grant\},6_i,5_o,true)} (s_1, 6) \xrightarrow{(\varnothing,\{grant\},4_i,6_o,false)} (s_0, 6)$, where we assumed that $\mathcal{D} = \mathbb{N}_0$, $d_0 = 0$, 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 Figure 2 satisfies the automaton from Figure 1.

2.3 Synthesis Problem

In this section, we define the model checking problem, bounded, and unbounded-but-finite synthesis problems. All the problems take as input a universal register automaton: one argument in favour of universal rather than non-deterministic automata is that the property "every data-request is eventually data-granted" can be expressed with a universal automaton, but not with a nondeterministic automaton.

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 [12, 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 [14], 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 [6], 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 [13], 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 [7] that allows you to fine-tune transition and output functions dependencies.

3 Solving the Bounded Synthesis Problem

Our approach is 5 points long.

(**1**) We start by defining a Boolean associate $A_{\mathbb{B}}$ of a universal co-Büchi register automaton A, which is a standard register-less universal co-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 Figures 1 and 2 as being standard, register-less, where *store* is a Boolean signal and has no special meaning. (**2**) We introduce a verifier automaton V, which tracks the equivalences between the registers $\mathscr{R}^A$ 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 "does A have a *rejecting* word?" 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 $A_{\mathbb{B}}@V$ is not suited for synthesis—we cannot synthesize from $A_{\mathbb{B}}@V$—for one of the two reasons: either we would have to allow the transducers to control the store actions of A, which brings unsoundness, or we would have to allow the environment to provide the input guards that do not correspond to any data-value, which brings incompleteness. (**3**) We add k_T fresh registers $\mathscr{R}^T$ to A that will be controlled by a transducer. To this end, we define the automaton T^{all}: it reads data-words *enhanced with store information of a transducer*, and filters out data-words that do not belong to any of the k_T-register transducers (e.g., data-words that have a value for o that was not seen before on i). We define $A \otimes T^{all}$, whose language is $L(A) \cap L(T^{all})^1$. (**4**) We enhance the Boolean associate $(A \otimes T^{all})_{\mathbb{B}}$ of $A \otimes T^{all}$ with information about equivalences between the registers $\mathscr{R}^T$ *and* $\mathscr{R}^A$; the resulting automaton is called $(A \otimes T^{all})_{\mathbb{B}}@W$, where W is a verifier similar to V but tailored towards synthesis. (**5**) Finally, we hide the information that should not be visible to a transducer, namely information related to the automaton registers $\mathscr{R}^A$. The resulting automaton is called $H = \mathit{hide}_A((A \otimes T^{all})_{\mathbb{B}}@W)$ and it is such that $\exists T : T \models A$ iff $\exists T_{\mathbb{B}} : T_{\mathbb{B}} \models H$. Furthermore, H, when viewed as a register automaton, is determinizable, and $L(H) \subseteq L(A)^1$.

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 $A = \langle P, \mathscr{P}, \mathscr{R}, \mathscr{d}_0, Q, q_0, \delta, F \rangle$, 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 $G_i = \{g_{i\mathscr{r}_1}, ..., g_{i\mathscr{r}_k}\}$, $G_o = \{g_{o\mathscr{r}_1}, ..., g_{o\mathscr{r}_k}\}$, $Asgn = \{a_{\mathscr{r}_1}, ..., a_{\mathscr{r}_k}\}$. Then:

[1] Actually, their alphabets differ, so this statement assumes A with extended alphabet.

- $P_{\mathbb{B}} = P \cup G_{\mathcal{i}} \cup G_{\mathcal{o}} \cup Asgn$,
- $\delta_{\mathbb{B}} : Q \times 2^{P_{\mathbb{B}}} \to 2^Q$ contains $(q, l \cup g_{\mathcal{i}} \cup g_{\mathcal{o}} \cup a, q') \in \delta_{\mathbb{B}}$ iff $(q, l, \bar{b}_{\mathcal{i}}, \bar{b}_{\mathcal{o}}, \bar{a}, q') \in \delta$, where $l \in 2^P$, $g_{\mathcal{i}} \in 2^{G_{\mathcal{i}}}$, $g_{\mathcal{o}} \in 2^{G_{\mathcal{o}}}$, $a \in 2^{Asgn}$, $\bar{b}_{\mathcal{i}} = (g_{\mathcal{i}\mathcal{r}_1} \in g_{\mathcal{i}}, ..., g_{\mathcal{i}\mathcal{r}_k} \in g_{\mathcal{i}}) \in \mathbb{B}^k$, $\bar{b}_{\mathcal{o}} = (g_{\mathcal{o}\mathcal{r}_1} \in g_{\mathcal{o}}, ..., g_{\mathcal{o}\mathcal{r}_k} \in g_{\mathcal{o}}) \in \mathbb{B}^k$, $\bar{a} = (a_{\mathcal{r}_1} \in a, ..., a_{\mathcal{r}_k} \in a) \in \mathbb{B}^k$. 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 $g_{\mathcal{i}} \in 2^{G_{\mathcal{i}}}$ *encodes* the guard $(g_{\mathcal{i}\mathcal{r}_1} \in g_{\mathcal{i}}, ..., g_{\mathcal{i}\mathcal{r}_k} \in g_{\mathcal{i}}) \in \mathbb{B}^k$, and vice versa; similarly for a letter from $2^{G_{\mathcal{o}}}$ and 2^{Asgn}.

A *Boolean path* is an infinite sequence $q_0 \xrightarrow{l_0 \cup g_{\mathcal{i}0} \cup g_{\mathcal{o}0} \cup a_0} q_1 \xrightarrow{l_1 \cup g_{\mathcal{i}1} \cup g_{\mathcal{o}1} \cup a_1} ...$ 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 $(q_0, \bar{\mathcal{d}}_0) \xrightarrow{(l_0, \mathcal{i}_0, \mathcal{o}_0, \bar{a}_0)} (q_1, \bar{\mathcal{d}}_1) \xrightarrow{(l_1, \mathcal{i}_1, \mathcal{o}_1, \bar{a}_1)} ...$ *corresponds* to a Boolean path $q_0 \xrightarrow{l_0 \cup g_{\mathcal{i}0} \cup g_{\mathcal{o}0} \cup a_0} q_1 \xrightarrow{l_1 \cup g_{\mathcal{i}1} \cup g_{\mathcal{o}1} \cup a_1} ...$ where $g_{\mathcal{i}j}$ encodes the guard $(\mathcal{i}_j = \bar{\mathcal{d}}_j[1], ..., \mathcal{i}_j = \bar{\mathcal{d}}_j[k])$, $g_{\mathcal{o}j}$ encodes the guard $(\mathcal{o}_j = \bar{\mathcal{d}}_j[1], ..., \mathcal{o}_j = \bar{\mathcal{d}}_j[k])$, 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 $\bar{a} = (\mathit{true}, \mathit{true})$, meaning that in a data-path the value of data-input is stored into the registers $\mathcal{r}_1$ and $\mathcal{r}_2$. Hence, in the next transition of the data-path, $\mathcal{i} = \mathcal{r}_1 \Leftrightarrow \mathcal{i} = \mathcal{r}_2$ must hold, but the Boolean path may have $g_{\mathcal{i}} = \{g_{\mathcal{i}\mathcal{r}_2}\}$ (describing the guard $\mathcal{i} \neq \mathcal{r}_1 \wedge \mathcal{i} = \mathcal{r}_2$). 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 $T = \langle I, O, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{d}_0, S, s_0, \tau \rangle$, 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: $I_{\mathbb{B}} = I \cup G_{\mathcal{i}}$, $G_{\mathcal{i}} = \{g_{\mathcal{i}\mathcal{r}_1}, ..., g_{\mathcal{i}\mathcal{r}_k}\}$, $O_{\mathbb{B}} = O \cup Asgn \cup O_k$, $Asgn = \{a_{\mathcal{r}_1}, ..., a_{\mathcal{r}_k}\}$, 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 $(s, l \cup g_{\mathcal{i}}, o \cup o_k \cup a, s')$ iff $(s, l, \bar{b}_{\mathcal{i}}, o, \tilde{o}_k, \bar{a}, s') \in \tau$ where $s, s' \in S$, $l \in 2^I$, $a \in 2^{Asgn}$ encodes $\bar{a} \in \mathbb{B}^k$, $g_{\mathcal{i}} \in 2^{G_{\mathcal{i}}}$ encodes $\bar{b}_{\mathcal{i}} \in \mathbb{B}^k$, and $o_k \in 2^{O_k}$ encodes $\tilde{o}_k \in [k]$. A

Boolean path is an infinite sequence $s_0 \overset{l_0 \cup g_{i0}, o_0 \cup o_{k0} \cup a_0}{\longrightarrow} s_1 \overset{l_1 \cup g_{i1}, o_1 \cup o_{k1} \cup a_1}{\longrightarrow}$ ... 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 $V_k = \langle P_V, \mathit{\Pi}, \pi_0, \delta_V \rangle$ where

- $\mathit{\Pi}$ is the set of all possible partitions of $\{\mathfrak{r}_1, ..., \mathfrak{r}_k\}$; the initial state $\pi_0 = \{\{\mathfrak{r}_1, ..., \mathfrak{r}_k\}\}$ contains the only partition. Later, we will a partition-state to track if the registers have the same value.
- $P_V = G_i \cup G_o \cup \mathit{Asgn}$ where $G_i = \{g_{i\mathfrak{r}_1}, ..., g_{i\mathfrak{r}_k}\}$, $G_o = \{g_{o\mathfrak{r}_1}, ..., g_{o\mathfrak{r}_k}\}$, $\mathit{Asgn} = \{a_{\mathfrak{r}_1}, ..., a_{\mathfrak{r}_k}\}$.
- $\delta_V : \mathit{\Pi} \times 2^{P_V} \to \mathit{\Pi}$ 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 $\mathfrak{r}_m = \mathfrak{r}_n$ of π (i.e., belonging to the same partition): $g_{i\mathfrak{r}_m} \in g_i \Leftrightarrow g_{i\mathfrak{r}_n} \in g_i$ and $g_{o\mathfrak{r}_m} \in g_o \Leftrightarrow g_{o\mathfrak{r}_n} \in g_o$;
    - $*$ for every $\mathfrak{r}_m \neq \mathfrak{r}_n$ of π (i.e., belonging to different partitions): $g_{i\mathfrak{r}_m} \in g_i \Rightarrow g_{i\mathfrak{r}_n} \notin g_i$ and $g_{o\mathfrak{r}_m} \in g_o \Rightarrow g_{o\mathfrak{r}_n} \notin g_o$;
  - the successor partition respects the assignment-letter a, formalized as follows. For every m, n in $[k]$, let e_{mn} denote that π contains $\mathfrak{r}_m = \mathfrak{r}_n$, and e'_{mn} is for π'. The value e'_{mn} is uniquely defined:

    $$e'_{mn} = (a_{\mathfrak{r}_m} \wedge a_{\mathfrak{r}_n}) \vee (\neg a_{\mathfrak{r}_m} \wedge a_{\mathfrak{r}_n} \wedge g_{i\mathfrak{r}_m}) \vee (a_{\mathfrak{r}_m} \wedge \neg a_{\mathfrak{r}_n} \wedge g_{i\mathfrak{r}_n}) \vee (\neg a_{\mathfrak{r}_m} \wedge \neg a_{\mathfrak{r}_n} \wedge e_{mn}).$$

    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 Figure 3.

$\mathbf{A_{\mathbb{B}}@V_k}$. Given a verifier $V_k = \left\langle P^V, Q^V, q_0^V, \delta^V \right\rangle$ and a register-less universal co-Büchi automaton $A_{\mathbb{B}} = \left\langle P^A, Q^A, q_0^A, \delta^A, F^A \right\rangle$, let $A_{\mathbb{B}}@V$ denote the universal co-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
- $F = Q^V \times F^A$.

Since $P^A = P' \cup G_i \cup G_o \cup \mathit{Asgn}$ (where P' are the Boolean signals of the register automaton A) and $P^V = G_i \cup G_o \cup \mathit{Asgn}$, the automaton $A_{\mathbb{B}}@V_k$ works

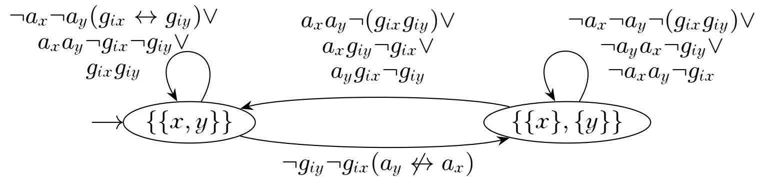
$\neg a_x \neg a_y (g_{ix} \leftrightarrow g_{iy}) \vee$
$a_x a_y \neg g_{ix} \neg g_{iy} \vee$
$g_{ix} g_{iy}$
$a_x a_y \neg (g_{ix} g_{iy}) \vee$
$a_x g_{iy} \neg g_{ix} \vee$
$a_y g_{ix} \neg g_{iy}$
$\neg a_x \neg a_y \neg (g_{ix} g_{iy}) \vee$
$\neg a_y a_x \neg g_{iy} \vee$
$\neg a_x a_y \neg g_{ix}$
$\{\{x, y\}\}$
$\{\{x\}, \{y\}\}$
$\neg g_{iy} \neg g_{ix} (a_y \not\leftrightarrow a_x)$

Fig. 3: A verifier automaton (a register-less deterministic looping automaton) for 2-register automata with $\mathscr{R} = \{x, y\}$. 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).

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*. Notice that falling out of the verifier component favours accepting; $L(A_{\mathbb{B}}@V_k) = \overline{L(V_k)} \cup L(A_{\mathbb{B}})$, or, equivalently, $\overline{L(AT_{\mathbb{B}}@V_k)} = L(V_k) \cap \overline{L(A_{\mathbb{B}})}$. Thus, the rejected words of $AT_{\mathbb{B}}@V_k$ are consistent and are rejected by $A_{\mathbb{B}}$.

Observation 2. *For every universal co-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.*

Proof. The first item follows from the definition of a data-pata. Consider the second item. Consider a Boolean path of $A_{\mathbb{B}}@V_k$

$$(q_0, \Pi_0) \xrightarrow{l_0 \cup g_{i0} \cup g_{o0} \cup a_0} (q_1, \Pi_1) \xrightarrow{l_1 \cup g_{i1} \cup g_{o1} \cup a_1} \ldots$$

(where q_j is a state of $A_{\mathbb{B}}$, Π_j is a state of V_k, $l_j \in 2^P$, $g_{ij} \in 2^{G_i}$, $g_{oj} \in 2^{G_o}$, and $a_j \in 2^{Asgn}$, for every $j \in \mathbb{N}_0$). We construct a corresponding data-path of A

$$(q_0, \bar{\mathscr{d}}_0) \xrightarrow{(l_0, i_0, o_0, \bar{a}_0)} (q_1, \bar{\mathscr{d}}_1) \xrightarrow{(l_1, i_1, o_1, \bar{a}_1)} \ldots :$$

- $\bar{\mathscr{d}}_0 = \mathscr{d}_0^k$;
- $\bar{a}_j \in \mathbb{B}^k$ encodes $a_j \in 2^{Asgn}$,
- $\bar{\mathscr{d}}_{j+1}$ is uniquely defined by $\bar{\mathscr{d}}_j$, i_j, and $\bar{a}_j$; and
- i_j and o_j are arbitrary such that $(\bar{\mathscr{d}}_j, i_j, o_j)$ satisfies the guards encoded by g_{ij} and g_{oj}. Such values exist, because Π_j and g_{ij} and g_{oj} are non-contradictory. Note that there are >1 possible values for i_j (in fact, infinitely many) iff g_{ij} encodes the guard $\bigwedge_{m \in [k]} i \neq \mathscr{r}_m$ (i.e., false^k); similarly for o_j.

□

The observation, together with the definition of acceptance by V_k, implies the following.

Corollary 1. *For every universal co-Büchi k-register automaton A:*
$A_{\mathbb{B}}@V_k$ has a rejected Boolean word $\Leftrightarrow$ A has a rejected data-word.

If we look at the dual automaton $\bar{A}$ (non-deterministic Büchi) and the dual $\overline{AT_{\mathbb{B}}@V_k}$, then the corollary states that non-emptiness of non-deterministic Büchi register automata is decidable. This result was earlier established in [12, Thm.1] using cutoffs (we discussed cutoffs on page 5). Our verifier uses a similar insight, but it is handy in the context of synthesis.

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

In the end, we will have a register-less automaton H, from which we will a Boolean associate of a register transducer. In the Boolean associate, the assignment actions are modelled as Boolean outputs. Therefore, the automaton H should have Boolean signals expressing the assignment actions of the Boolean transducer. The automaton T^{all} fulfills this purpose: it adds k_T fresh registers to A that will be controlled by transducers via fresh Boolean signals.

$\mathbf{T}^{\boldsymbol{all}}$. Let $k_T \in \mathbb{N}$ and let $Asgn^T = \{a_{\mathscr{r}^T_1}, ..., a_{\mathscr{r}^T_{k_T}}\}$ be fresh Boolean signals. T^{all} is a deterministic co-Büchi k_T-register automaton $\langle P, \mathscr{P}, \mathscr{R}, \mathscr{d}_0, Q, q_0, \delta, F\rangle$ with $P = I \cup O \cup Asgn^T$, $\mathscr{P} = \{\mathscr{i}, \mathscr{o}\}$, $Q = \{q_0, \lightning\}$, $F = \{\lightning\}$. The transition function

$$Q \times 2^{I \cup O \cup Asgn^T} \times \mathbb{B}^{k_T}_{\mathscr{i}} \times \mathbb{B}^{k_T}_{\mathscr{o}} \to Q \times \mathbb{B}^{k_T}$$

- for every $\bar{g}_i \in \mathbb{B}^{k_T}_{\mathscr{i}}$, $\bar{g}_o \in \{\bar{g} \in \mathbb{B}^{k_T} \mid \exists j. \bar{g}[j] = \mathit{true}\}$, and $a \in 2^{Asgn^T}$, contains $(q_0, \bar{a})$ where $\bar{a}[j] = \mathit{true}$ iff $a_{\mathscr{r}^T_j} \in a$ for every $j \in [k_T]$;
- when $\bar{g}_o$ does not satisfy the above condition, it transits from q_0 to $\lightning$;
- it self-loops in $\lightning$ without storing for every letter.

In words: T^{all} ensures that the value of data-output $\mathscr{o}$ comes from a register and the assignment actions are *synced* with the Boolean signals $Asgn^T$.

Observation 3. *Let $k_T \in \mathbb{N}$, then: for every $w \in (2^{I \cup O \cup Asgn^T} \times \mathscr{D}^2)^\omega$:*

$$w \models T^{all} \;\Leftrightarrow\; \exists T\colon w \in L(T),$$

where T is a k_T-register transducer (possibly, $|S| = \infty$) whose output is extended with $Asgn^T$ signals that are synced with T's assignment actions.

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 that transducers are deterministic). That is a minor technical detail though.

$\mathbf{A} \otimes \mathbf{T}^{\boldsymbol{all}}$. The product $A \otimes T^{all}$ of a universal co-Büchi register automaton $A = \left\langle P^A, \mathscr{P}, \mathscr{R}^A, \mathscr{d}_0, Q^A, q^A_0, \delta^A, F^A \right\rangle$ and $T^{all} = \left\langle P^T, \mathscr{P}, \mathscr{R}^T, \mathscr{d}_0, Q^T, q^T_0, \delta^T, F^T \right\rangle$, where $P^T = P^A \cup Asgn^T$, is a universal co-Büchi $(k_A + k_T)$-register automaton $\langle P, \mathscr{P}, \mathscr{R}, \mathscr{d}_0, Q, q_0, \delta, F\rangle$, where $P = P^T$, $\mathscr{R} = \mathscr{R}^A \,\dot{\cup}\, \mathscr{R}^T$, $Q = Q^A \times Q^T$, $q_0 = (q^A_0, q^T_0)$, $F = F^A \times Q^T \cup Q^A \times F^T$, and the transition function

$$\delta : Q \times 2^{I \cup O \cup Asgn^T} \times \mathbb{B}^{k_A + k_T}_{\mathscr{i}} \times \mathbb{B}^{k_A + k_T}_{\mathscr{o}} \to 2^{Q \times \mathbb{B}^{k_A}} \times \mathbb{B}^{k_T}$$

respects both δ^A and δ^T.

Observation 4. *For every $k_T \in \mathbb{N}$, universal co-Büchi k_A-register automaton A, and $w \in (2^{P^A \cup Asgn^T} \times \mathcal{D}^2)^\omega$:*

$$w \models A \otimes T^{all} \;\Leftrightarrow\; w \models T^{all} \text{ and } w|_{2^{P^A}} \models A,$$

where $w|_{2^{P^A}}$ is a projection of w into 2^{P^A}.

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 $AT_{\mathbb{B}}@W$ that will be 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 Section 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 $Asgn^T = \{a_{\mathscr{r}_1^T}, ..., a_{\mathscr{r}_{k_T}^T}\}$, $G_i^T = \{g_{i\mathscr{r}_1^T}, ..., g_{i\mathscr{r}_{k_T}^T}\}$, and O_{k_T} has enough Boolean signals to encode the numbers $[k_T]$. For synthesis we want our target specification automaton to have the same alphabet. The automaton $AT_{\mathbb{B}}@V_k$ uses o-guards instead of signals O_k, 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 $P' = (P \setminus (G_o^T \cup G_o^A)) \cup O_{k_T}$ 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 $AT_{\mathbb{B}}$, $i \in 2^I$, $o \in 2^O$, $g_i \in 2^{G_i^A \cup G_i^T}$, $g_o \in 2^{G_o^A \cup G_o^T}$, $a \in 2^{Asgn^A \cup Asgn^T}$):

- let $J = \{j_1, ..., j_l\} \subset \mathbb{N}$ be such that g_o contains $o = \mathscr{r}_j^T$ 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 $j \in [k_T]$.
- Note that if J is empty (g_o requires that $\bigwedge_{t \in [k_T]} o \neq \mathscr{r}_t^T$), then we do not add transitions to δ', because no transducer can produce such a value for o.

Notice that $AT_{\mathbb{B}}@W$, just like $AT_{\mathbb{B}}@V_k$, accepts inconsistent words (those fall out of the original V_k). Inconsistency in those words can come from signals $G_i^A \cup G_i^T$. Later, these Boolean signals will either be hidden (G_i^A) or under environment control (G_i^T), which means that a transducer will not be able to sabotage the specification by producing inconsistent words.

The following observation resembles Observation 2, but focuses on k_T-register transducers.

Observation 5. *For every universal co-Büchi k_A-register automaton A, $k_T \in \mathbb{N}$:*

- *every data-path of $A \otimes T^{all}$ has exactly one corresponding Boolean path in $AT_{\mathbb{B}}@W$;*
- *every Boolean path of $AT_{\mathbb{B}}@W$ has either one or infinitely many corresponding data-paths in $A \otimes T^{all}$.*

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

We cannot use $AT_{\mathbb{B}}@W$ for synthesis, because it uses Boolean signals that are not visible to transducers (underlined): $I \cup O \cup \underline{G_i^A} \cup G_i^T \cup \underline{G_o^A} \cup O_{k_T} \cup \underline{Asgn^A} \cup Asgn^T$. 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 universal co-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

$$\delta' : Q \times 2^I \times 2^O \times 2^{G_i^T} \times 2^{O_{k_T}} \times 2^{Asgn^T} \to 2^Q$$

consists of transitions $q \xrightarrow{(i,o,g_i^T,j,a^T)} Q'$ that satisfy the following: 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 universal co-Büchi register automaton A, $k_T \in \mathbb{N}$:*

- *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 $AT_{\mathbb{B}}@W$.*

Proof. The first item follows from the definition of $hide_A(AT_{\mathbb{B}}@W)$.

Consider the second item. Fix a path $p = q_1 \overset{\sigma_1}{\to} q_2 \overset{\sigma_2}{\to} \ldots$ of $hide_A(AT_{\mathbb{B}}@W)$. By definition, for every transition $q_j \overset{\sigma_j}{\to} q_{j+1}$ of $hide_A(AT_{\mathbb{B}}@W)$, there must be some transition $q_j \overset{\sigma'_j}{\to} q_{j+1}$ of $AT_{\mathbb{B}}@W$, where σ'_j and σ_j agree on the values of shared signals. Hence, in order to get the desired path of $AT_{\mathbb{B}}@W$, we do the following: for every j, *arbitrary* choose $\sigma'_j \in 2^{I \cup O \cup G_i^A \cup G_i^T \cup G_o^A \cup O_{k_T} \cup Asgn^A \cup Asgn^T}$ that satisfies $\delta_{AT_{\mathbb{B}}@W}$ and agrees with $\sigma_j \in 2^{I \cup O \cup G_i^T \cup O_{k_T} \cup Asgn^T}$ on the values of shared signals. □

Lemma 1. *For every k_T-register transducer T and universal co-Büchi k_A-register automaton A:*

$$\big(\exists w \in L(T) : w \not\models A\big) \;\Leftrightarrow\; \big(\exists w_{\mathbb{B}} \in L(T_{\mathbb{B}}) : w_{\mathbb{B}} \not\models hide_A(AT_{\mathbb{B}}@W)\big).$$

Proof. Both directions follow from the definitions and Observations 5 and 6.

Consider direction $\Leftarrow$. The word $w_{\mathbb{B}} \in (2^{I \cup O \cup G_i^T \cup O_{k_T} \cup Asgn^T})^\omega$ induces a path $\pi_{t_b} \in (S \times 2^{I \cup O \cup G_i^T \cup O_{k_T} \cup Asgn^T})^\omega$ on $T_{\mathbb{B}}$ and a rejected path $\pi_h \in (Q_h \times 2^{I \cup O \cup G_i^T \cup O_{k_T} \cup Asgn^T})^\omega$ on $hide_A(AT_{\mathbb{B}}@W)$. By Observation 6, π_h corresponds to at least one path $\pi_{atw} \in (Q_h \times 2^{I \cup O \times G_i^A \cup G_i^T \cup G_o^A \cup O_{k_T} \cup Asgn^A \cup Asgn^T})^\omega$ of $AT_{\mathbb{B}}@W$. By Observation 5, π_{atw} corresponds to at least one data-path $\pi_{at} \in (Q_{at} \times 2^{I \cup O \cup Asgn^T} \times \mathcal{D}^2)^\omega$ of $A \otimes T^{all}$, which is rejected by A, because π_h is rejected by $A_{\mathbb{B}}$. Thus, we get $w \in (2^{I \cup O} \times \mathcal{D}^2)^\omega$ from π_{at} by projecting, which completes the direction. Notice that a data-path $\pi_t \in (S \times 2^{I \cup O \cup Asgn^T} \times \mathcal{D}^2)^\omega$ of T induced by w corresponds to the Boolean path π_{t_b} of $T_{\mathbb{B}}$ induced by $w_{\mathbb{B}}$, despite the particular choices of π_{atw} and π_{at}.

The other direction is similar. □

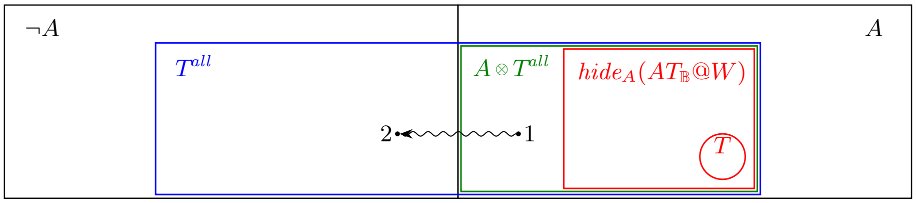

Fig. 4: Inclusion between languages. The automaton $hide_A(AT_{\mathbb{B}}@W)$ is Boolean, but here it is viewed as a register automaton. Also, the alphabet of A is extended with $Asgn^T$ to coincide with that of $A \otimes T^{all}$ and $hide_A(AT_{\mathbb{B}}@W)$. Figure 5 justifies the existence of point 1, which explains why $hide_A(AT_{\mathbb{B}}@W)$ can be a *strict* subset of $A \otimes T^{all}$. The snake line indicates "for every T: if it has point 1, then it also has point 2" (by Lemma 1). Thus, if $T \models A$ for some k_T-register transducer, then it must be located inside $hide_A(AT_{\mathbb{B}}@W)$.

The lemma implies a solution to the bounded synthesis problem.

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

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

where T is a k_T-register transducer.

The right side of the theorem (the standard Boolean synthesis problem) holds iff it holds for finite-state transducers (e.g., see [15]). 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.*

Let us consider the complexity of our approach. The automaton $hide_A(AT_{\mathbb{B}}@W)$ has $|Q_A| \cdot |\Pi|$ states, where Q_A is the number of states in A and $|\Pi|$ is the number of partitions of the set $\{1, ..., k\}$ where $k = k_T + k_A$. The latter is a Bell number [16] and is less than $(\frac{0.792k}{\ln(k+1)})^k$ [2, Thm 2.1]. Hence the number of states in $hide_A(AT_{\mathbb{B}}@W)$ is less than $|Q_A| \cdot (\frac{0.792k}{\ln(k+1)})^k$, and the complexity of our approach is in $synth(|Q_A| \cdot (\frac{0.792k}{\ln(k+1)})^k)$, where $synth(m) = 2^{c \cdot m}$ is the complexity of synthesis from a universal co-Büchi automaton with m states [15, Thm.2] (c is a constant). This is an upper bound, the lower bound is open, thus we get:

Corollary 3. *The bounded synthesis problem can be solved in $2^{c \cdot |Q_A| \cdot (\frac{0.792k}{\ln(k+1)})^k}$ time, where $k = k_A + k_T$, $|Q_A|$ and k_A is the number of states and registers in a given universal automaton, and c is a constant.*

Finally, Figure 4 depicts the relation between the languages of utilized automata. It shows that the approach makes use of determinizable subset of $A \otimes T^{all}$.

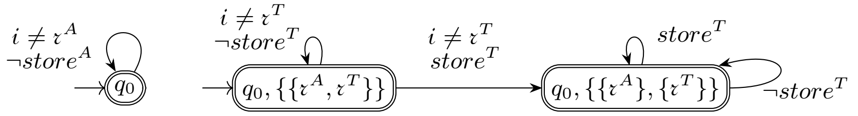

Fig. 5: Universal co-Büchi register automata to show the existence of point 1 in Fig.4. On the left is 1-register automaton A: it accepts the words where at some moment the signal i equals to $\mathscr{d}_0$ (and no restrictions on the values of o). On the right is $hide_A(AT_{\mathbb{B}}@W)$ where $k_T = 1$: when viewed as a register automaton, it accepts the words where the first value of i is $\mathscr{d}_0$ (plus some restrictions on o). Hence, $L(hide_A(AT_{\mathbb{B}}@W)) \subsetneq L(A \otimes T^{all})$. 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 a first-order linear temporal logic with equality, LTL(EQ)[1] and its variants ∃LTL(EQ) and ∀LTL(EQ), known as IPTL in [17] and VLTL in [11]. Then Section 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 Section 4.3. Then Section 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 (no complete conversion can exist). The latter automata are consumed by our synthesizer.

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

4.1 LTL(EQ) (also known as IPTL [17] and VLTL [11])

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 $k \in \mathbb{N}$):

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

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 $\neg\,\mathsf{F}\,\varphi$, $\mathsf{F}\,\varphi = true\,\mathsf{U}\,\varphi$, $\varphi_1 \vee \varphi_2$ 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 $\mathscr{d}_1, ..., \mathscr{d}_k \in \mathcal{D}$ either $cond[x_1 \leftarrow \mathscr{d}_1, ..., x_k \leftarrow \mathscr{d}_k]$ does not hold or $w \models \varphi[x_1 \leftarrow \mathscr{d}_1, ..., x_k \leftarrow \mathscr{d}_k]$;
- $w \models \exists x_1 ... x_k . cond . \varphi$ iff there exists $\mathscr{d}_1, ..., \mathscr{d}_k \in \mathcal{D}$ such that $cond[x_1 \leftarrow \mathscr{d}_1, ..., x_k \leftarrow \mathscr{d}_k]$ holds and $w \models \varphi[x_1 \leftarrow \mathscr{d}_1, ..., x_k \leftarrow \mathscr{d}_k]$;
- let ϕ have the same grammar as φ except that instead of data-variables it has data-values; then

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

- $w \models \mathit{true}$;
- $w \not\models \phi$ 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 $\mathscr{d} \in \mathscr{D}$, $w \models i = \mathscr{d}$ iff in w_1 the data-proposition i has the value $\mathscr{d}$; 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 [10].

A *k-register-guessing automaton* is a tuple $A = \langle P, \mathscr{P}, \mathscr{R}, Q, q_0, \delta, F, E\rangle$ (notice: no initial register value $\mathscr{d}_0$ and a new element E) with transition function δ of the form $Q \times 2^P \times \mathbb{B}^k_i \times \mathbb{B}^k_o \to 2^Q$ (notice: no assignment component on the right), where $E \subseteq \mathscr{R} \times \mathscr{R}$ 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

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

An accepting word is defined as for register automata.

4.3 Converting $\exists$LTL(EQ) into Register-Guessing Automata

This section describes the conversion of $\exists$LTL(EQ) formulas into register-guessing automata with the same language. The fact that a conversion is possible was noted in [8, Sec.4], however they did not describe the conversion itself.

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

($w_{\mathbb{B}}$) Given a word $w \in (2^P \times \mathscr{D}^2)^\omega$ and $x_1, ..., x_k \in \mathscr{D}$, let $w_{\mathbb{B}} \in (2^P \times \mathbb{B}^k_i \times \mathbb{B}^k_o)^\omega$ be the word derived from w by replacing every value of i and o in w by the vectors of Boolean 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 a new literal $g_{i\mathscr{r}_i}$ and every expression $o = x_i$ with $g_{o\mathscr{r}_i}$. This introduces $2k$ new Boolean propositions, let $P_{\mathbb{B}} = P \cup \{g_{i\mathscr{r}_1}, ..., g_{i\mathscr{r}_k}\} \cup \{g_{o\mathscr{r}_1}, ..., g_{o\mathscr{r}_k}\}$. Let $\varphi_{\mathbb{B}}(g_{i\mathscr{r}_1}, ..., g_{i\mathscr{r}_k}, g_{o\mathscr{r}_1}, ..., g_{o\mathscr{r}_k})$ be the resulting LTL formula over Boolean propositions $P_{\mathbb{B}}$.

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

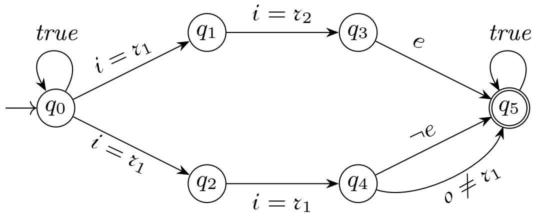

Fig. 6: A 2-register-guessing automaton: $P = \{e\}$, $\mathscr{R} = \{r_1, r_2\}$, $E = \{(r_1, r_2)\}$. The edges have symbolic labels, e.g., the edge labeled with $i = r_1$ encodes 16 edges, for different valuations of e, $i = r_2$, $o = r_1$, and $o = r_2$.

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*.

For example, the automaton in Figure 6 expresses the formula

$$\neg\forall x_1 \neq x_2.\, \mathsf{G}\begin{bmatrix} i = x_1 \wedge \mathsf{X}\, i = x_2 \rightarrow \mathsf{XX}\,\neg e \\ i = x_1 \wedge \mathsf{X}\, i = x_1 \rightarrow \mathsf{XX}(e \wedge o = x_1)\end{bmatrix}$$

that says: compare the data-input i at two consecutive points and then (i) whenever they are equal, raise e and output the data, (ii) otherwise, lower e.

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

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 $Q \times 2^P \times G \rightarrow 2^Q$ and $Q \times 2^P \times G \rightarrow 2^{Q \times \mathbb{B}^k}$, (previously we had $\mathbb{B}_i^k \times \mathbb{B}_o^k$ instead of G), where $g \in G$ has the form $g = true \mid g \wedge g \mid i \sim r \mid o \sim r$ where $\sim$ denotes $=$ or $\neq$, and $r \in \mathscr{R}$. 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 $A' = \langle P, \mathscr{P}, \mathscr{R}, d_0, Q', q_0', \delta', F'\rangle$ (**conversion-2**):

- $Q' = Q \times \mathbb{B}^k$. The Boolean component encodes, for every $\mathscr{r}_i \in \mathcal{R}$, whether the register $\mathscr{r}_i$ 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 $\mathscr{r}_i$ with $b_i = \mathit{false}$ *uninitialized.*
- $F' = \{(q, b_1, ..., b_k) \in Q' \mid q \in F\}$.
- 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 $\mathscr{i} \neq \mathscr{r}_i$ or $\mathscr{o} \sim \mathscr{r}_i$, then abort. Because the register $\mathscr{r}_i$ is uninitalized ($b_i = \mathit{false}$), we cannot know the valuation of $\mathscr{i} \neq \mathscr{r}_i$ or $\mathscr{o} \neq \mathscr{r}_i$. In contrast, if the guard g contains $\mathscr{i} = \mathscr{r}_i$, we can assume that it holds and store $\mathscr{i}$ into $\mathscr{r}_i$ (we cannot do this for $\mathscr{o} = \mathscr{r}_i$, because the automata do not allow for storing $\mathscr{o}$).
    - ∗ 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 $\mathscr{i} = \mathscr{r}_i$.
      - · The action a stores $\mathscr{i}$ into $\mathscr{r}_i$ iff g contains $\mathscr{i} = \mathscr{r}_i$ and $b_i = \mathit{false}$.
      - · The guard g' contains $\mathscr{i} \sim \mathscr{r}_i$ iff g contains $\mathscr{i} \sim \mathscr{r}_i$ and $b_i = \mathit{true}$; similarly for $\mathscr{o} \sim \mathscr{r}_i$.
    - ∗ Finally, we account for the inequality set E and update g' as follows. For every $(\mathscr{r}_i, \mathscr{r}_j) \in E$: if $b_i = \mathit{true}$ and the action a contains $\mathscr{r}_j = \mathscr{i}$, then add to g' the expression $\mathscr{i} \neq r_i$.
      (Here we assume that the A-transition is not contradictory, namely, it is not the case that $\exists(\mathscr{r}_i, \mathscr{r}_j) \in E : b_i = \mathit{false} \wedge b_j = \mathit{false} \wedge (\mathscr{i} = \mathscr{r}_i) \in g \wedge (\mathscr{i} = \mathscr{r}_j) \in g$. Such transitions cannot be executed in A and can be removed beforehand.)
- Note that the automaton A' never compares $\mathscr{i}$ nor $\mathscr{o}$ with a register that was uninitialized. Therefore, the component $\mathscr{d}_0$ of A' can be anything from $\mathcal{D}$.

The automaton A' has $|Q'| = |Q|{\cdot}2^k$, but the number of reachable states is $|Q| \cdot k$, because every A'-transition $(q, b_1, ..., b_k) \xrightarrow{(l,g,a)} (q', b'_1, ..., b'_k)$ satisfies $(b'_1, ..., b'_k) \geq (b_1, ..., b_k)$.

An example of the conversion is in Figure 7.

Observation 8. *Given a register-guessing automaton A. If conversion-2 succeeds and produces a register automaton A', then $L(A) = L(A')$,*

Proof. We need to prove that $\forall w \in (2^P \times \mathcal{D}^2)^\omega : w \models A \Leftrightarrow w \models A'$.

Consider the direction $\Rightarrow$. The acceptance $w \models A$ means that there exists $\mathcal{R}_0 \in \mathcal{D}^k$ and an accepting data-path p starting in configuration $(q_0, \mathcal{R}_0)$ and corresponding to p the accepting Boolean path $p_\mathbb{B}$:

$$p = (q_0, \mathcal{R}_0) \xrightarrow{(l_0,\mathscr{i}_0,\mathscr{o}_0)} (q_1, \mathcal{R}_0) \xrightarrow{(l_1,\mathscr{i}_1,\mathscr{o}_1)} ...$$

$$p_\mathbb{B} = q_0 \xrightarrow{(l_0,g_0)} q_1 \xrightarrow{(l_1,g_1)} ...$$

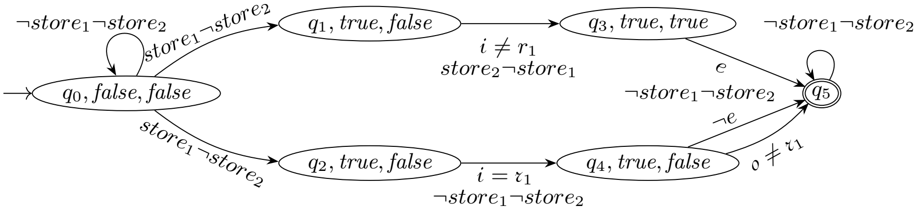

Fig. 7: A 2-register automaton converted from the register-guessing automaton in Figure 6.

(for every $j \in \mathbb{N}_0$: $l_j \in 2^P$, $i_j \in \mathcal{D}$, $o_j \in \mathcal{D}$, and $g_j \in G$). We build inductively the accepting data-path p' of A' (and corresponding to p' the Boolean path $p'_{\mathbb{B}}$):

$$p' = \big((q_0, B_0), \mathcal{R}'_0\big) \xrightarrow{(l_0,i_0,o_0,a'_0)} \big((q_1, B_1), \mathcal{R}'_1\big) \xrightarrow{(l_1,i_1,o_1,a'_1)} ...$$

$$p'_{\mathbb{B}} = (q_0, B_0) \xrightarrow{(l_0,g'_0,a'_0)} (q_1, B_1) \xrightarrow{(l_1,g'_1,a'_1)} ...$$

($\forall j \in \mathbb{N}_0$: $B_j \in \mathbb{B}^k$, $a'_j \in \mathbb{B}^k$, $\mathcal{R}_j \in \mathcal{D}^k$, and i_j, o_j, l_j are as for p) as follows.

- The path p' starts with $B_0 = \mathit{false}^k$ and $\mathcal{R}_0 = d_0^k$.
- The construction of A' uniquely defines a'_j, g'_j, and B_{j+1} from B_j and g_j.
- The value of $\mathcal{R}'_{j+1}$ is uniquely defined by $\mathcal{R}'_j$, a'_j, and i.
- By the construction, every g'_j is less restricting or equal to g_j, and we never compare i or o with uninitialized registers. Because $(i_j, o_j, \mathcal{R}_0) \models g_j$, we have that $(i_j, o_j, \mathcal{R}'_j) \models g'_j$, for every $j \in \mathbb{N}_0$. Hence, every transition of p' is indeed a transition of A', and p' is indeed a path of A'.

The direction $\Leftarrow$ is similar to the above. The data-path p and corresponding to p the Boolean path $p_{\mathbb{B}}$ of A are uniquely constructed from a given data-path p' and corresponding to p' the Boolean path $p'_{\mathbb{B}}$ of A'. When proving that p is indeed a path of A, we use the property of A' that it never compares i nor o with a register whose value was not written before. □

Combined together, the conversions give us the following.

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

5 Conclusion

In this paper we introduced a sound and complete approach to synthesis of register transducers from specifications given as register automata. Although we focused on automata with the co-Büchi acceptance, others (e.g., parity) looks doable. The approach works (incompletely) for specifications given as quantified temporal logic formulas, by converting them into register automata. In particular, we investigated the two directions—richer automata and suitable temporal logic—raised by Ehlers et al.[6, Sect.6].

We are working on extending the approach to automata with guards that, in addition to =, have operators >, +, and on the question of decidability of the unbounded-but-finite synthesis problem that is open. It would be interesting to combine our approach with the approach to synthesis of reactive programs [9]. It would also be interesting to do a synthesis case study, possibly for specifications with costs.

Acknowledgements. We thank Orna Kupferman for comments on the early draft and helpful discussions. This work was supported by the Austrian Science Fund (FWF) under the RiSE National Research Network (S11406) and by the Hebrew University.

References

1. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB) (2016), www.SMT-LIB.org
2. Berend, D., Tassa, T.: Improved bounds on bell numbers and on moments of sums of random variables. Probability and Mathematical Statistics 30(2), 185–205 (2010)
3. Church, A.: Logic, arithmetic, and automata. In: International Congress of Mathematicians (Stockholm, 1962), pp. 23–35. Institute Mittag-Leffler, Djursholm (1963)
4. Demri, S., D'Souza, D., Gascon, R.: Temporal logics of repeating values. J. Log. and Comput. 22(5), 1059–1096 (Oct 2012), http://dx.doi.org/10.1093/logcom/exr013
5. Demri, S., Lazić, R.: Ltl with the freeze quantifier and register automata. ACM Trans. Comput. Logic 10(3), 16:1–16:30 (Apr 2009), http://doi.acm.org/10.1145/1507244.1507246
6. Ehlers, R., Seshia, S.A., Kress-Gazit, H.: Synthesis with identifiers. In: International Conference on Verification, Model Checking, and Abstract Interpretation. pp. 415–433. Springer (2014)
7. Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT 15(5-6), 519–539 (2013)
8. 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.) NASA Formal Methods. pp. 1–18. Springer (2017)
9. Gerstacker, C., Klein, F., Finkbeiner, B.: Bounded Synthesis of Reactive Programs. ArXiv e-prints (Jul 2018), to appear at ATVA'18
10. Grumberg, O., Kupferman, O., Sheinvald, S.: Variable automata over infinite alphabets. In: International Conference on Language and Automata Theory and Applications. pp. 561–572. Springer (2010)
11. Grumberg, O., Kupferman, O., Sheinvald, S.: Model checking systems and specifications with parameterized atomic propositions. In: International Symposium on Automated Technology for Verification and Analysis. pp. 122–136. Springer (2012)
12. Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science 134(2), 329 – 363 (1994), http://www.sciencedirect.com/science/article/pii/0304397594902429
13. Kupferman, O., Vardi, M.: Synthesis with incomplete informatio. In: 2nd International Conference on Temporal Logic. pp. 91–106. Manchester (July 1997)
14. Lazić, R., Nowak, D.: A unifying approach to data-independence. In: Palamidessi, C. (ed.) CONCUR 2000 — Concurrency Theory. pp. 581–596. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
15. 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, Austin, Texas, USA, January 11-13, 1989. pp. 179–190. ACM Press (1989), http://doi.acm.org/10.1145/75277.75293

16. Wikipedia contributors: Bell number — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Bell_number&oldid=832584649 (2018), [Online; accessed 8-August-2018]
17. 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), http://doi.acm.org/10.1145/512644.512661

A Change History

- Aug 14, 2018: fixed a small bug in Section 3.4.
- Aug 9, 2018: rewriting into using universal instead of non-deterministic automata. Added complexity result.
- Aug 2, 2018: the extended version of the final version submitted to ATVA.