Register-Bounded Synthesis

Ayrat Khalimov
School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel
ayrat.khalimov@gmail.com

Orna Kupferman
School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel
orna@cs.huji.ac.il

Abstract

Traditional synthesis algorithms return, given a specification over finite sets of input and output Boolean variables, a finite-state transducer all whose computations satisfy the specification. Many real-life systems have an infinite state space. In particular, behaviors of systems with a finite control yet variables that range over infinite domains, are specified by automata with infinite alphabets. A register automaton has a finite set of registers, and its transitions are based on a comparison of the letters in the input with these stored in its registers. Unfortunately, reasoning about register automata is complex. In particular, the synthesis problem for specifications given by register automata, where the goal is to generate correct register transducers, is undecidable.

We study the synthesis problem for systems with a bounded number of registers. Formally, the register-bounded realizability problem is to decide, given a specification register automaton $A$ over infinite input and output alphabets and numbers $k_s$ and $k_e$ of registers, whether there is a system transducer $T$ with at most $k_s$ registers such that for all environment transducers $T'$ with at most $k_e$ registers, the computation $T||T'$, generated by the interaction of $T$ with $T'$, satisfies the specification $A$. The register-bounded synthesis problem is to construct such a transducer $T$, if exists.

The bounded setting captures better real-life scenarios where bounds on the systems and/or its environment are known. In addition, the bounds are the key to new synthesis algorithms, and, as recently shown in [24], they lead to decidability. Our contributions include a stronger specification formalism (universal register parity automata), simpler algorithms, which enable a clean complexity analysis, a study of settings in which both the system and the environment are bounded, and a study of the theoretical aspects of the setting: in particular, the differences among a fixed, finite, and infinite number of registers, and the determinacy of the corresponding games.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Synthesis, Register Automata, Register Transducers

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.25


1 Introduction

Synthesis is the automated construction of a system from its specification. The specification distinguishes between outputs, generated by the system, and inputs, generated by its environment. The system should realize the specification, namely satisfy it against all possible environments. Thus, for every sequence of inputs, the system should generate a sequence of outputs so that the induced computation satisfies the specification [10, 30]. The systems are modelled by transducers: automata whose transitions are labeled by letters from the input alphabet, which trigger the transition, and letters from the output alphabet, which are generated when the transition is taken. Since its introduction, synthesis has been one of the most studied problems in formal methods, with extensive research on wider settings, heuristics, and applications [25, 1].

© Ayrat Khalimov and Orna Kupferman; licensed under Creative Commons License CC-BY

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 25; pp. 25:1–25:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
Until recently, all studies of the synthesis problem considered finite state transducers that realize specifications given by temporal-logic formulas over a finite set of Boolean propositions or by finite-state automata. Many real-life systems, however, have an infinite state space. One class of infinite-state systems, motivating this work, consists of systems in which the control is finite and the source of infinity is the domain of the variables in the systems. This includes, for example, data-independent programs [37, 20, 27], software with integer parameters [5], communication protocols with message parameters [11], datalog systems with infinite data domain [4, 36], and more [8, 6]. Lifting automata-based methods to the setting of such systems requires the introduction of automata with infinite alphabets. The latter include registers [33], pebbles [28, 34], or variables [18, 19], or handle the infinite alphabets by attributing it by labels from an auxiliary finite alphabet [3, 2].

A register automaton [33] has a finite set of registers, each of which may contain a letter from the infinite alphabet. The transitions of a register automaton do not refer explicitly to each of the (infinitely many) input letters. Rather, they compare the letter in the input with the content of the registers, and may also store the input letter in a register. Several variants of this model have been studied. For example, [21] forces the content of the registers to be different, [28] adds alternation and two-wayness, [22] allows the registers to change their content nondeterministically during the run, and [35] adds the ability to check for uniqueness of the input letter. Likewise, register transducers are adjusted to model systems whose interaction involves input and output variables over an infinite domain: their transitions are labeled by guards that compare the value in the input with the content of the registers. In addition, while taking a transition, the transducer stores this value in some of its registers and outputs a value stored in one of its registers. For example, a transition of a register transducer can be “in state $q_5$, if the value in the input is not equal to the value stored in register $\#1$, then store the value in the input into register $\#2$, output the value stored in register $\#1$, and transit to state $q_3$”. A register automaton can thus specify properties like “every value read in the input in two successive cycles is output in the next cycle”. For more elaborated examples, see Examples 1 and 2.

The transition to infinite alphabets makes reasoning much more complex. In particular, the universality and containment problems for register automata are undecidable [28], and so is the synthesis problem for specifications given by register automata [14]. While the specifications used for the undecidability result in [14] are register automata with a fixed number of registers, the realizing transducers are equipped with an unbounded queue of registers: they can push the inputs into the queue, and later compare the inputs with the values in the queue. This, for example, is helpful for realizing specifications like “every value that appears in the input has to eventually appear on the output twice”. While the latter can be specified by a register automaton with a single register, a realizing transducer for it may behave as follows: it queues every incoming value into its queue, outputs the value stored in the head of the queue twice, and dequeues it – which requires an unbounded queue of registers. Moreover, as shown in [15], the synthesis problem stays undecidable even when the number of registers in the realizing transducer is finite, yet not known in advance. In [24], it is shown that bounding the number of registers of the realizing transducer makes the synthesis problem decidable. Essentially, such a bound enables an abstraction of the infinite number of register valuations to a finite number of equivalence relations. In more details, since the transitions of the specification register automaton only compare the value in the input with the content of its registers, we can abstract the exact values stored in the registers and only maintain their partition into equivalence classes: two registers are in the same class if they agree on the values stored in them. In particular, such a partition fixes the transition that the automaton should take, and can be updated whenever the input value is stored in some register.
In this paper we offer a comprehensive study of the synthesis problem for systems with a bounded number of registers. As has been the case with bounded synthesis in the finite-state setting [31, 13, 16, 26], the motivation for the study is both conceptual and computational: First, the bounded setting captures better real-life scenarios where bounds on the systems and/or its environment are known. Second, the bounds are the key to new synthesis algorithms, and in the case of systems with an infinite variable domain, they lead to decidability. Note that the only parameter we bound is the number of registers. In particular, the size of the alphabet stays infinite, and the size of the system and its environment stays unbounded\(^1\).

Let us start with the conceptual motivation. It is by now realized that requiring a realizing system to satisfy the specification against all possible environments is often too demanding. Dually, allowing all possible systems is perhaps not demanding enough. This issue is traditionally approached by adding assumptions on the system and/or the environment, which are modeled as part of the specification (see e.g. [9]). In bounded synthesis in the finite-state setting, the assumptions on the system and its environment are given by means of bounds on the sizes of their state space [31, 26]. In the setting of register transducers, bounding the size of the state spaces of the system and its environment is not of much interest, as a register may be used to store the value of the state. Thus, the interesting parameter to bound is the number of allowed registers. Indeed, this setting corresponds to systems with a finite control and a finite number of memory elements, each maintaining a value from an infinite domain. Formally, the \textit{register-bounded realizability problem} is to decide, given a specification register automaton \(A\) over infinite input and output alphabets and numbers \(k_s\) and \(k_e\) of registers, whether there is a system transducer \(T\) with at most \(k_s\) registers such that for all environment transducers \(T'\) with at most \(k_e\) registers, the computation \(T \parallel T'\), generated by the interaction of \(T\) with \(T'\), satisfies the specification \(A\). The \textit{register-bounded synthesis problem} is to construct such a transducer \(T\), if exists.

We continue to the computational motivation and describe our contribution. Our specifications are given by \textit{universal register parity automata on infinite words} (reg-UPW, for short). Thus, each configuration of the automaton may have several successor configurations, and an infinite word is accepted if all the possible runs on it are accepting. Reg-UPWs are more expressive than deterministic register parity automata or universal register Büchi automata, and are more succinct than universal register co-Büchi automata. Reg-UPWs are incomparable with nondeterministic register parity automata (reg-NPW). There are good reasons to work with the universal (rather than nondeterministic) model. First, basic questions are undecidable for reg-NPW. In particular, [12] shows undecidability of the universality problem for nondeterministic register weak automata with a single register, which can be shown to imply undecidability of reg-NPW register-bounded synthesis. Second, as we demonstrate in Section 2, the class of properties that are expressible by reg-UPWs is more interesting in practice. In particular, reg-UPWs are easily closed under conjunction, which is crucial for synthesis.

We describe a simple algorithm for the register-bounded synthesis problem for reg-UPW specifications ([24] only handles co-Büchi automata), which enables a clean complexity analysis ([24] only shows decidability). We study the settings in which both the system and

\(^1\) We note, however, that bounding the number of states in the realizing transducer has proven to be helpful also in the context of systems over infinite alphabets. For example, [17] describes a CEGAR-based synthesis algorithm that approaches the general undecidable synthesis problem by iteratively refining under-approximating systems of bounded sizes.
the environment are bounded ([24] only bounds the system), and we study the theoretical aspects of the setting; in particular, the differences between a fixed, a finite yet unbounded, and an infinite number of registers, and the determinacy of the corresponding games.

Our synthesis algorithm reduces the register-bounded synthesis problem to the traditional synthesis problem. Specifically, given a specification reg-UPW $A$ with $k_A$ registers, and numbers $k_s$ and $k_e$, we construct a (register-less) UPW $A'$ that abstracts the values in the registers of $A$ and consider instead equivalences among registers in the three sets of registers involved: these of $A$, and these of the system and environment transducers. The synthesis problem for $A$ is then reduced to that of $A'$. In Section 3 we solve the case where the environment is not bounded (thus $k_e = \infty$) and then in Section 4 continue to the general case. Our complexity analysis carefully takes into account the fact that in the determinization of $A'$, the registers of $A$ and the environment behave universally, whereas these of the system behave deterministically. Accordingly, the complexity of the register-bounded synthesis problem for $A$ with $n$ states, finite alphabet of size $m$, and index $c$, can be solved in time $(c m n(k_s + k_e + k_A))^{O(c n(k_s + k_e + k_A)^{k_s + k_e + k_A})}$. Thus, it is polynomial in $m$, exponential in $c$, $n$, and $k_s$, and doubly-exponential only in $k_A$ and $k_e$. In the full version [23], we also study determinacy of register-bounded synthesis and show that for all $k_s \in \mathbb{N}$ and $k_e \in \mathbb{N} \cup \{\infty\}$, the problem is not determined: there are specifications that are neither realizable by a bounded system (with respect to bounded environments), nor their negations are realizable by a bounded environment (with respect to bounded systems). This corresponds to the picture obtained for bounded synthesis for finite-state systems, where the size of the state space is bounded (we bound only the number of registers) [26]. We also examine the difference in the strength of systems and environments with a fixed, finite, or infinite number of registers, and the existence of a cut-off point, namely a finite-model property characterizing settings where a finite and bounded number of registers suffices.

2 Preliminaries

2.1 Register Automata

Let $\Sigma_I$ and $\Sigma_O$ be two finite alphabets and let $D$ be an infinite domain of data values. We consider systems that get inputs in $\Sigma_I \times D$ and respond with outputs in $\Sigma_O \times D$. Let $\Sigma = \Sigma_I \times \Sigma_O$. Computations of systems as above are words in $(\Sigma \times D)^\omega$. Register automata specify languages of such words. Let $B = \{true, false\}$. A $k$-register word automaton is a tuple $A = \langle \Sigma, Q, q_0, R, v_0, \delta, \alpha \rangle$, where $\Sigma$ is a finite alphabet, $Q$ is the set of states, $q_0 \in Q$ is an initial state, $R$ is a set of $k$ registers, $v_0 \in D^R$ is an initial register valuation, $\delta : Q \times (\Sigma \times \mathbb{B}^R \times \mathbb{B}^R) \rightarrow 2^{Q \times \mathbb{B}^R}$ is a transition function, and $\alpha$ is an acceptance condition (we later define several acceptance conditions). Intuitively, when $A$ is in state $q$ and reads a letter $\sigma \in \Sigma \times D$, it compares $i$ and $o$ with the content of its registers and branches into several new configurations according to the result of this comparison. In more detail, rather than specifying a transition for each element in $\Sigma \times D \times D$, the transition function $\delta$ specifies a transition for each element in $\Sigma \times D \times D$, where the two guards in $\mathbb{B}^R$ compare the values stored in the registers with $i$ and $o$. Then, $\delta$ directs $A$ into a set of pairs in $Q \times \mathbb{B}^R$, each describing a successor state and a storing mask, indicating which registers are going to store $i$.

A configuration of $A$ is a pair $\langle q, v \rangle \in Q \times D^R$, describing the state that $A$ visits and the content of its registers. A run of $A$ starts in the configuration $\langle q_0, v_0 \rangle$, and continues to form an infinite sequence of successive configurations. In order to define runs formally, we first need some notations. Given a valuation $v \in D^R$ and a value $d \in D$, let $v \sim d$ denote the
Boolean assignment $g \in \mathbb{B}^R$ that indicates the agreement of $v$ with $\delta$. Thus, for every $r \in R$, we have $g(r) = true$ if $v(r) = \delta$. The function $update : \mathbb{D}^R \times \mathbb{D} \times \mathbb{B}^R \rightarrow \mathbb{D}^R$ maps a valuation $v \in \mathbb{D}^R$, a value $\delta \in \mathbb{D}$, and a storing mask $a \in \mathbb{B}^R$, to the valuation obtained from $v$ by changing the value stored in registers that are positive in $a$ to $\delta$. Formally, for every $r \in R$, we have that $update(v, \delta, a)(r)$ is $\delta$ if $a(r) = true$ and is $v(r)$ otherwise. Note that it need not be the case that $update(v, \delta, a) \sim \delta = a$. Indeed, if $v(r) = \delta$, then $update(v, \delta, a)(r) = \delta$ regardless of $a(r)$.

For two configurations $\langle q', o' \rangle$ and $\langle q, o \rangle$ in $Q \times \mathbb{D}^R$, and a triple $\langle \sigma, i, o \rangle \in \Sigma \times \mathbb{D} \times \mathbb{D}$, we say that $\langle q', o' \rangle$ is a $\langle \sigma, i, o \rangle$-successor of $\langle q, o \rangle$ if there exists $a \in \mathbb{B}^R$ such that $\langle q', a \rangle \in \delta(q, \langle \sigma, i, o \sim i, v \sim o \rangle)$ and $o' = update(v, i, a)$.

Now, a run of $A$ on a word $w = \langle \sigma_0, i_0, o_0 \rangle \langle \sigma_1, i_1, o_1 \rangle \ldots \in (\Sigma \times \mathbb{D} \times \mathbb{D})^\omega$ is an infinite sequence $\langle q_0, o_0 \rangle \langle q_1, o_1 \rangle \ldots \in (Q \times \mathbb{D}^R)^\omega$ of configurations such that for every $j \geq 0$, we have that $\langle q_j, o_j \rangle$ is a $\langle \sigma_j, i_j, o_j \rangle$-successor of $\langle q_j, o_j \rangle$. Note that there may be several different runs on the same word. Note also that since $\delta$ may return an empty set of possible transitions, a configuration $\langle q_j, o_j \rangle$ need not have $\langle \sigma_j, i_j, o_j \rangle$-successors. There, the sequence of successive configurations is finite, and is not a run.

When $A$ is a parity automaton, $\alpha : Q \rightarrow \{0, \ldots, c - 1\}$, for an index $c \in \mathbb{N}$, a run $\rho$ is accepting if the maximal rank that is visited by $\rho$ infinitely often is even. Formally, $\rho = \langle q_0, o_0 \rangle \langle q_1, o_1 \rangle \ldots$ is accepting if max $\{j \in \{0, \ldots, c - 1\} : \alpha(q_j) = j \text{ for infinitely many } l \geq 0\}$ is even. The co-Büchi acceptance condition is a special case of parity, with $c = 2$. Thus, $\rho$ is accepting if vertices $\langle q, o \rangle$ with $\alpha(q) = 1$ are visited only finitely often. When $A$ is universal, it accepts the word $w$ if all the runs of $A$ on $w$ are accepting. Note that since we require runs to be infinite, the universal quantification on the runs means that a configuration with no successors is like an accepting configuration: once we reach it, there are no restrictions on the suffix of the word. The language of $A$, denoted $L(A)$, is the set of all words that $A$ accepts. We sometimes use $w \models A$ to indicate that $w \in L(A)$. We use reg-UPW and reg-UCW to abbreviate a universal register parity and co-Büchi automata, respectively. A (register-less) UPW can be viewed as a special case of a reg-UPW with no registers. In particular, it has no initial valuation and its transition function is of the form $\delta : Q \times \Sigma \rightarrow 2^Q$.

▶ Example 1. The reg-UCW $A$ appearing in Figure 1 specifies an arbiter with a single output signal $ack$ (that is, $\Sigma_I$ is a singleton, and we ignore it, and $\Sigma_O = 2^{\{ack\}}$) that gets in each moment in time an input data value $i$, and outputs either $ack$ or $\neg ack$ along with an output data value $o$. It accepts a word if every input data value different from the previous one is eventually outputted with $ack$. The acceptance condition $\alpha$ requires runs to visit $q_1$ only finitely often. The reg-UCW $A$ has a single register, thus $R = \{r_1\}$, and we describe vectors in $\Sigma \times \mathbb{B}^R \times \mathbb{B}^R$ by triples in $\{ack, \neg ack\} \times \{0, 1\} \times \{0, 1\}$, possibly replacing some of the parameters by $\_\_$, indicating that both values of this parameter apply. We continue to describe

$\begin{align*}
i = r_1 \\
i \neq r_1/store_1 \\
ack \lor o \neq r_1 \\
q_0 \\
i \neq r_1/store_1 \\
q_1
\end{align*}$

Figure 1 The reg-UCW $A$. The edge labels are symbolic, where the expressions $i \neq r_1$ and $i = r_1$ mean that the $i$-guard is 0 and 1 respectively, and the expression $o \neq r_1$ means that the $o$-guard is 0. The label $store_1$ means the storing mask is 1, while its absence means it is 0. The state $q_1$ is doubly-circled, indicating that a run is accepting iff it visits $q_1$ only finitely often.
the transition function. First, \( \delta(q_0, (\_ , 1, \_)) = \{(q_0, 0)\} \). That is, if the input data value agrees with the one stored in \( r_1 \), we only loop in \( q_0 \). Then, \( \delta(q_0, (\_ , 0, \_)) = \{(q_0, 1), (q_1, 1)\} \). That is, if the input data value differs from the one stored in \( r_1 \), then \( A \) both loops in \( q_0 \) and sends a copy to \( q_1 \), and stores the value of the input data value in \( r_1 \). In state \( q_1 \), we have \( \delta(q_1, (\text{ack}, \_ , 1)) = \emptyset \), thus the copy sent to \( q_1 \) fulfills its mission when it reads an ack with an output data value that agrees with the one stored in \( r_1 \). In all other cases, the copy stays in \( q_1 \). Thus, \( \delta(q_1, (\neg\text{ack}, \_ , 1)) = \delta(q_1, (\text{ack}, \_ , 0)) = (q_1, 0) \). The parity acceptance condition \( \alpha = \{q_0 \mapsto 0, q_1 \mapsto 1\} \) then guarantees that all copies sent to \( q_1 \) eventually fulfill their missions. We note that the universality of \( A \) is used in order to detect all data values that are not stored in \( r_1 \): a copy of the automaton is launched for each of them. Such a detection is impossible in a deterministic or even a nondeterministic register automaton.

### 2.2 Register Transducers

Register transducers model systems with inputs in \( \Sigma_I \times D \) and outputs in \( \Sigma_O \times D \). Every such system implements a strategy \( (\Sigma_I \times D)^+ \rightarrow \Sigma_O \times D \), describing the output it generates after reading a sequence of inputs. A register transducer is a tuple \( T = (\Sigma_I, \Sigma_O, S, s_0, R, \tau) \), where \( \Sigma_I \) and \( \Sigma_O \) are input and output finite alphabets, \( S \) is a set of states, \( s_0 \in S \) is an initial state, \( R \) is a set of registers, \( v_0 \in D^R \) is an initial register valuation, and \( \tau : S \times (\Sigma_I \times B^R) \rightarrow S \times B^R \times \Sigma_O \times R \) is a transition function. Intuitively, when \( T \) is in state \( s \) and reads a letter \( (i, i) \in \Sigma_I \times D \), it compares \( i \) with the content of its registers. Depending on \( i \) and the comparison, it transits deterministically to a successor state and may store the data value \( i \) into its registers. It also outputs a letter in \( \Sigma_O \) and a value stored in one of the registers. Note that a register may store either its initial value or some value seen earlier as a data input.

Formally, a configuration of \( T \) is a pair in \( S \times D^R \), and successive configurations are defined in a way similar to the one defined for automata, except that \( T \) is deterministic: given a configuration \( (s, v) \in S \times D^R \), and an input \( (i, i) \in \Sigma_I \times D \), let \( \tau(s, (i, v \sim i)) = (s', a, o, r) \). Then, the \( (i, i) \)-successor of \( (s, v) \) is \( (s', \text{update}(v, i, a)) \).

Given an input word \( w = (i_0, v_0)(i_1, v_1)\ldots \in (\Sigma_I \times D)\omega \), the \textit{run} of \( T \) on \( w \) is the sequence \( (s_0, v_0)(s_1, v_1)\ldots \in (S \times D^R)\omega \), where for all \( j \geq 0 \), we have that \( (s_j, v_j) \) is the \( (i_j, i_j) \)-successor of \( (s_j, v_j) \). For every \( j \geq 0 \), let \( \tau(s_j, i_j, v_j \sim i_j) = (s_{j+1}, a_j, o_j, r_j) \). Then, the \textit{computation of \( T \)} on \( w \) is the sequence \( (i_0, o_0)(i_0, o_0)(i_1, o_1, i_1, o_1)\ldots \in ((\Sigma_I \times \Sigma_O) \times D)\omega \) such that for every \( j \geq 0 \), we have that \( a_j = \text{update}(v_j, i_j, a_j)(r_j) \). Thus, the transducer moves from \( s_j \) to \( s_{j+1} \), stores \( i_j \) in registers that are positive in \( a_j \), and then outputs \( o_j \) and the (updated) content of register \( r_j \). A (register-less) transducer is a special case of a register transducer with no registers. In particular, it has no initial valuation and its transition function is of the form \( \tau : S \times \Sigma_I \rightarrow S \times \Sigma_O \).

For a register transducer \( T \) and a reg-UPW \( A \), we say that \( T \) realizes \( A \), denoted \( T \models A \), if for all input words \( w \in (\Sigma_I \times D)^\omega \), the computation of \( T \) on \( w \) is in the language of \( A \).

**Example 2.** Figure 2 describes a register transducer that realizes the reg-UCW from Example 1. The input alphabet \( \Sigma_I \) is a singleton and we ignore it. The output alphabet \( \Sigma_O = 2^{\{\text{ack}\}} \), and the register set \( R = \{r_1, r_2\} \). The transducer loops in the initial state \( s_0 \) if the current data input equals the previous data input (which is stored in register \( r_1 \)). Otherwise \( (i \neq r_1) \), the transducer stores the new data value into \( r_1 \), does not raise \( \text{ack} \), outputs the value of register \( r_1 \) (it has to output something), and moves into state \( s_1 \). Now, if it does not see a new data input \( (i = r_1) \), then – in order to acknowledge the previous data input – it raises \( \text{ack} \), outputs the previous data input from \( r_1 \), and returns into \( s_0 \).
Alternatively, if in state $s_1$ the transducer sees a new data input ($i \neq r_1$), then it stores into $r_2$, raises $\text{ack}$, outputs the previous data input from $r_1$, and moves into $s_2$. From there, if no new data input was seen, the transducer moves into $s_3$, while outputting the value of $r_2$ and raising $\text{ack}$. And so on. Thus, in states $s_0$ and $s_1$ register $r_1$ contains the previous data input, while in states $s_2$ and $s_3$ it is stored in register $r_2$. Finally, register $r_1$ is initialized with the same value as the automaton register, while $r_2$ can start with anything. We conclude with a remark that there is a simpler transducer that realizes the same reg-UCW: It always raises $\text{ack}$, stores alternatingly into $r_1$ and $r_2$ while outputting alternatingly the value of $r_2$ and $r_1$. But such a transducer produces spurious $\text{acks}$, while our transducer does not.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{A register transducer that realizes the reg-UCW $A$ from Example 1. The edge labeling for $\Sigma_D$ and the guards is symbolic, and is similar to that in Figure 1.}
\end{figure}

2.3 Synthesis with an Infinite or Unbounded Number of System Registers

The realizability problem is to decide, given a reg-UPW $A$ over $\Sigma_I \times \Sigma_O \times D \times D$, whether there is a register transducer all whose computations are accepted by $A$. The synthesis problem is to construct such a transducer, if exists.

The realizability and synthesis problems in the context of specifications and systems with an infinite data domain was first studied in [14]. The transducers in [14] have an infinite number of registers, all initialized to the same value. The automata in [14] are universal register automata with a variant of weak acceptance condition, and additionally do not allow for register re-assignment. It is shown in [14] that the synthesis problems is undecidable, already for automata with only two registers. Since our automata and transducers are more powerful, undecidability applies to our setting. Thus, when the number of registers in the system is infinite, the realizability and synthesis problems are undecidable.

Consider now the case where the number of registers is finite but not fixed a-priori. It is shown in [12] that the nonemptiness problem for universal 2-register automata on finite words is undecidable. It is not hard to reduce their nonemptiness problem to the synthesis problem for 2-register UPWs, which implies the undecidability of the latter. Thus, we get the following.

\begin{itemize}
\item \textbf{Theorem 3 ([12, 14])}. The synthesis problem of transducers with an infinite or a finite but unbounded number of registers for specifications given by 2-register UPWs is undecidable. In the case of an infinite number of registers, undecidability holds even when the transducer registers are initialized with the same value.
\end{itemize}

3 Synthesis with a Fixed Number of System Registers

The system-bounded realizability problem is to decide, given a reg-UPW $A$ over $\Sigma_I \times \Sigma_O \times D \times D$ and a number $k_s$ of registers, whether there is a transducer with at most $k_s$ registers all whose computations are accepted by $A$. The system-bounded synthesis problem is to construct such a transducer, if exists.
Let $A = \langle \Sigma, Q, q_0, R_A, v^A_0, \delta, \alpha \rangle$, and let $|R_A| = k_A$. Recall that $\Sigma = \Sigma_I \times \Sigma_O$. We define a UPW $A'$ (that is, with no registers) that abstracts the values stored in $R_A$. Instead, $A'$ maintains an equivalence relation over the registers of $A$ and the registers of the realizing transducer, indicating which of them agree on the values stored in them.

Let $R_s$ denote a set of $k_s$ registers, namely those of the realizing transducer (we subscript its elements by $s$ as this transducer models the system), and let $R = R_A \cup R_s$. For valuations $v_A \in \mathbb{D}^{R_+}$ and $v_s \in \mathbb{D}^{R_\times}$, let $v_A \cup v_s$ be the valuation in $\mathbb{D}^R$ obtained by taking their union. Likewise, for a valuation $v \in \mathbb{D}^R$, let $v_A$ and $v_s$ denote the projections of $v$ on $R_A$ and $R_s$, respectively. Let $\Pi$ be the set of all equivalence relations over $R$. Consider an element $\pi \in \Pi$, thus $\pi \subseteq R \times R$. For two registers $r, r' \in R$, we write $\pi(r, r')$ to denote that $r$ and $r'$ are equivalent in $\pi$. Note that $r$ and $r'$ may be both in $R_A$, both in $R_s$, or one in $R_A$ and one in $R_s$. Each equivalence relation $\pi \in \Pi$ induces a partition of $R$ into equivalence classes, and we sometimes refer to the elements in $\Pi$ as partitions of $R$. Then, for $\pi \in \Pi$, we talk about sets $S \in \pi$, where $S \subseteq R$, and $\pi(r, r')$ indicates that $r$ and $r'$ are in the same set in the partition. Let $f : \mathbb{D}^R \to \Pi$ map a register valuation $v \in \mathbb{D}^R$ to the partition $\pi \in \Pi$, where for every two registers $r, r' \in R$, we have that $\pi(r, r')$ if $v(r) = v(r')$.

Recall that we describe guards and storing masks on a set $R$ of registers by Boolean functions in $\mathbb{B}^R$. Each assignment $g \in \mathbb{B}^R$ corresponds to a set of registers characterized by $g$. In the sequel, we sometimes refer to Boolean assignments as sets, thus assume that $g \subseteq R$, and talk about union and intersection of assignments, referring to the sets they characterize. Consider a partition $\pi$ of $R$ and a Boolean assignment $g_s \subseteq R_s$. We say that $g_s$ is $\pi$-consistent if there is an equivalence class $S \in \pi \cup \{\emptyset\}$ such that $S \cap R_s = g_s$. We then say that $(\pi, g_s)$ chooses $S$. Note that for $g_s = \emptyset$, the set $S$ is either empty or contains no system registers, and might be not unique. For example, if $R_A = \{s_1, s_2, s_3, s_4\}$, $R_s = \{s_5, s_6\}$, and $\pi = \{\{s_1\}, \{s_2, s_3\}, \{s_4, s_5\}, \{s_6\}\}$, then $(\pi, \{s_5\})$ chooses only $\{s_4, s_5\}$, the pair $(\pi, \{s_6\})$ chooses only $\{s_6\}$, and $(\pi, \emptyset)$ chooses $\{s_1\}, \{s_2, s_3\},$ or $\emptyset$. For a set $S_A \subseteq R_A$, we say that $(\pi, g_s)$ $A$-chooses $S_A$ if there is a set $S \in \pi \cup \{\emptyset\}$ such that $(\pi, g_s)$ chooses $S$ and $S_A = S \cap R_A$. Thus, $(\pi, g_s)$ $A$-chooses $S_A$ if $(\pi, g_s)$ chooses a set whose $R_A$ registers are these in $S_A$. Continuing the previous example, $(\pi, \{s_5\})$ $A$-chooses $\{s_4\}$, the pair $(\pi, \{s_6\})$ $A$-chooses $\emptyset$, and $(\pi, \emptyset)$ $A$-chooses $\{s_1\}, \{s_2, s_3\},$ or $\emptyset$. Finally, for a register $r \in R$, the pair $(\pi, r)$ $A$-chooses the unique set $S_A \subseteq R_A$ if $S_A = S \cap R_A$, for the set $S \in \pi$ such that $r \in S$. In the example above, the pairs $(\pi, \emptyset)$ and $(\pi, \emptyset)$ both $A$-choose $\{s_4\}$, and the pair $(\pi, \emptyset)$ $A$-chooses $\emptyset$.

The following lemma follows immediately from the definitions.

**Lemma 4.** Consider a partition $\pi$ of $R = R_A \cup R_s$ and a valuation $v \in \mathbb{D}^R$ s.t. $f(v) = \pi$. Then:

(a) for every $i \in D$, the guard $v_s \sim i$ is $\pi$-consistent and $A$-chooses the guard $v_A \sim i$,

(b) for every guard $g \in (\pi \cup \{\emptyset\})$, there exists $i \in D$ satisfying ($v \sim i$) = $g$, and

(c) for every $r \in R$, the pair $(\pi, r)$ $A$-chooses $v_A \sim v(r)$.

Recall the function $update : \mathbb{D}^R \times D \times \mathbb{B}^R \to \mathbb{D}^R$, where $update(g, d, a)$ is obtained from $v$ by storing $d$ in the registers in $a$. We now define a function $update' : \Pi \times \mathbb{B}^R \times \mathbb{B}^R \to \Pi$, which adjusts the update function to the abstraction of valuations by partitions. Intuitively, for a partition $\pi \in \Pi$, a guard $g \in (\pi \cup \{\emptyset\})$, and a storing mask $a \subseteq R$, we obtain the partition $update'(\pi, g, a)$ from $\pi$ by moving the registers in $a$ either into the equivalence class of $g$ (if $g$ is not empty), or into a new equivalence class. Formally, $update'(\pi, g, a) = \{S \setminus a : S \in \pi \setminus g \} \cup \{g \cup a\}$. Note that, in particular, $update'(\pi, \emptyset, a) = \{S \setminus a : S \in \pi \} \cup \{a\}$.
Lemma 5. For every valuation $v \in \mathcal{D}^R$, value $i \in \mathcal{D}$, and storing mask $a \subseteq R$, we have that $f(update(v, i, a)) = update'(f(v), v \sim i, a)$.

We are now ready to define the abstraction of $A$. In addition to $k_v$, the abstraction is parameterized by a partition $\pi_0$ of the system and automaton registers. Given $k_v$ and $\pi_0$, the $(k_v, \pi_0)$-abstraction of $A$ is the UPW $A' = (\Sigma', Q', q'_0, \delta', \alpha')$ with the following components.

- $Q' = Q \times \Pi$ and $q'_0 = (q_0, \pi_0)$. Thus, each state in $A'$ is a pair $\langle q, \pi \rangle$, abstracting configurations $\langle q, v_A \rangle$ of $A$ and register valuations $v_s$ of an anticipated transducer that satisfy $f(v_s) = \pi$.
- $\Sigma' = \Sigma \times \mathcal{B}^R \times \mathcal{A} \times \mathcal{B}^R$. Recall that in $A$, the transition function is $\delta : Q \times (\Sigma \times \mathcal{B}^R \times \mathcal{A} \times \mathcal{B}^R) \rightarrow 2Q \times \mathcal{B}^R$, and when $A$ is in configuration $\langle q, v_A \rangle$ and reads a letter $\langle \sigma, i, o \rangle \in \Sigma \times \mathcal{D} \times \mathcal{D}$, it proceeds according to $\langle \sigma, g^A_i, g^A \rangle \in \Sigma \times \mathcal{B}^R \times \mathcal{B}^R$, where $g^A_i$ is $v_A \sim i$ and $g^A$ is $v_A \sim o$. Also, each successor state $q'$ is paired with a storing mask $a^A \in \mathcal{B}^R$, which induces a successor configuration $\langle q', update(v, i, a^A) \rangle$. Intuitively, each letter $\langle \sigma, g^A_i, r_s, a^A \rangle \in \Sigma'$, together with the current partition, induces choices for $(\langle \sigma, g^A_i, r_s, a^A \rangle \in \Sigma \times \mathcal{B}^R \times \mathcal{B}^R \times \mathcal{B}^R)$ which determine the transitions in $A$ that the abstraction follows.

- For every state $\langle q, \pi \rangle \in Q'$ and letter $\langle \sigma, g^A_i, r_s, a^A \rangle \in \Sigma'$, we have that $\langle q', \pi' \rangle \in \delta'(\langle q, \pi \rangle, \langle \sigma, g^A_i, r_s, a^A \rangle)$ iff there exist $g^A_i, g^A, a^A \in \mathcal{B}^R$ such that the following conditions hold.
  - $g^A_i$ is $A$-chosen by $\langle \pi, g^A \rangle$. Let $g_i = g^A_i \cup g^A$. Note that $g_i \in (\pi \cup \{\varnothing\})$.
  - Recall that the output value in register transducers refers to the updated register values, namely their values in the successor configuration. Therefore, when we compare the data output of a transducer with the register values of the automaton, we first have to update the values of the system transducer. For this, we introduce the partition $\pi^*$. Let $\pi^*$ be the partition after updating the system registers in $\pi$ according to the guard $g^A_i$ and the storing mask $a^A_i$. Thus, $\pi^* = update'(\pi, g^A_i, a^A_i)$.
  - $g^A_i$ is $A$-chosen by $\langle \pi^*, r_s \rangle$. Note that since the set chosen by $\langle \pi^*, r_s \rangle$ is not empty, $g^A_i$ is unique.
  - $\langle q', a^A_i \rangle \in \delta(q, \langle \sigma, g^A_i, g^A \rangle)$.

- We can now complete updating the partition. The partition $\pi'$ is the result of updating the registers of $A$ in $\pi^*$ according to the guard $g^A_i$ and the storing mask $a^A_i$. Let $g_i' = g_i \cup a^A_i$ be the updated guard after system storing. Then $\pi' = update'(\pi^*, g_i', a^A_i)$.

- The acceptance condition of $A'$ is induced from the one of $A$. Thus, for every state $\langle q, \pi \rangle \in Q'$, we have that $\alpha'(\langle q, \pi \rangle) = \alpha(q)$.

Recall that the abstraction of $A$ is parameterized by both the number of registers that the system transducer may have as well as an initial partition for the registers of both the system and the automaton. Let $v_A \in \mathcal{D}^R$ be a valuation of the automaton registers. A partition $\pi \in \Pi$ is consistent with $v_A$ if there is a register valuation $v_s \in \mathcal{D}^R$ such that $\pi = f(v_A \cup v_s)$. Thus, all automaton registers are related according to $v_A$, and the system registers are unrestricted.

Example 6. Let $\mathcal{D} = \mathbb{N}$, $R_A = \{\ast 1, \ast 2, \ast 3, \ast 4\}$, and $R_s = \{\ast 5, \ast 6, \ast 7\}$. Then the partition $\pi = \{\{\ast 1, \ast 4, \ast 5\}, \{\ast 2, \ast 6\}, \{\ast 3\}, \{\ast 7\}\}$ is consistent with the valuation $v_A \in \mathcal{D}^R$ for which $v_A(\ast 1) = v_A(\ast 4) = 9$, $v_A(\ast 2) = 2$, and $v_A(\ast 3) = 13$. Indeed, taking $v_s \in \mathcal{D}^R$ with $v_s(\ast 5) = 9$, $v_s(\ast 6) = 2$, and $v_s(\ast 7) = 14$ results in $\pi = f(v_A \cup v_s)$. Note that different valuations $v_s \in \mathcal{D}^R$ may witness the consistency of $\pi$ with $v_A$. In our example, all these with $v_s(\ast 5) = 9$, $v_s(\ast 6) = 2$, and $v_s(\ast 7) \notin \{2, 9, 13\}$. Also, several different partitions may be consistent with a given valuation $v_A \in \mathcal{D}^R$. In our example, all these in which register $\ast 1$ and $\ast 4$ are in the same set, different from the (different) sets of $\ast 2$ and $\ast 3$.
We can now state our main theorem, relating the realizability of $A$ with realizability of its abstraction. Consider a $k_s$-register $\Sigma_I/\Sigma_O$-transducer $T = (\Sigma_I, \Sigma_O, S, s_0, R, v_0, \tau)$. We can view $T$ as a (register-less) $\Sigma_I/\Sigma_O'$-transducer $T'$, for $\Sigma'_I = \Sigma_I \times \mathbb{B}^R$ and $\Sigma'_O = \mathbb{B}^R \times \Sigma_O \times R$. Indeed, the transition function $\tau : S \times (\Sigma_I \times \mathbb{B}^R) \rightarrow S \times \mathbb{B}^R \times \Sigma_O \times R$ of $T$ can be viewed as $\tau' : S \times \Sigma'_I \rightarrow S \times \Sigma'_O$. When $v_0 \in \mathbb{D}^{R_s}$ is fixed, we say that $T$ and $T'$ correspond to each other. Essentially, our main theorem follows from the fact that a reg-UPW $A$ is realized by a $k_s$-transducer $T$ iff the abstraction of $A$ is realized by the register-less transducer that corresponds to $T$. Formally, we have the following.

**Theorem 7.** Consider a reg-UPW $A$ with $\Sigma = \Sigma_I \times \Sigma_O$, set of registers $R_A$, and an initial valuation $v_0^A$. Then, $A$ is realizable by a $k_s$-register $\Sigma_I/\Sigma_O$-transducer if there is a partition $\pi_0$ of $R = R_s \cup R_A$, consistent with $v_0^A$, such that the $(k_s, \pi_0)$-abstraction of $A$ is realizable by a $\Sigma_I \times (\mathbb{B}^{R_s} / (\Sigma_O \times R_s \times \mathbb{B}^{R_s}))$-transducer. In particular, a transducer that realizes the $(k_s, \pi_0)$-abstraction of $A$ corresponds to a $k_s$-register transducer that realizes $A$.

**Proof sketch.** Let $A = (\Sigma, \rho, q_0, R_A, v_0^A, \delta, \alpha)$ and let $A'$ be its $(k_s, \pi_0)$-abstraction, where $\pi_0$ is a partition of $R$ consistent with $v_0^A$. We prove that for every valuation $v_0^A \in \mathbb{D}^{R_s}$, satisfying $f(v_0^A) = \pi_0$, a $k_s$-register $\Sigma_I/\Sigma_O$-transducer $T$ initialized with $v_0^A$, and registerless $\Sigma_I \times (\mathbb{B}^{R_s} / (\mathbb{B}^{R_s} \times \Sigma_O \times R_s))$-transducer $T'$, where $T$ and $T'$ correspond to each other, it holds that $T \models A$ iff $T' \models A'$. The theorem then follows.

Assume first that $T \not\models A$. We prove that $T' \not\models A'$. Since $T \not\models A$, there is an input sequence $w_T = \langle i_0, i_0 \rangle, \ldots, a run $p_T = \langle s_0, \rho_0 \rangle$ such that $T$ generates when it follows $p_T$, and a rejecting run $\rho_A = \langle q_0, \rho_0 \rangle$ of $A$ on the computation $w_T$. Note that $A$ may have several runs on $w_T$. Since $T$ is universal, and $A$ rejects $w_T$, we know that at least one of them does not satisfy $\alpha$. We show that $w_T$ and $p_T$ induce an input sequence $w_T'$, to $T'$ such that $A'$ rejects the computation of $T'$ on $w_T'$. We define $w_T' = \langle i_0, v_0^A, i_1, v_1^A \rangle$. The word $w_T'$ uniquely defines the computation $w_T'$ and the run $p_T = s_0, s_1, \ldots$ of $T'$. We now define the rejecting run $\rho_{A'}$ of $A'$ on $w_T'$. It starts in the configuration $\langle q_0, \pi_0 \rangle$. Suppose that in step $j \geq 0$, the run $\rho_{A'}$ reaches the configuration $\langle q, v_A \rangle$, the run $p_T$ reaches the configuration $\langle s, v_s \rangle$, and the run $\rho_{A'}$ reaches the state $\langle q, \pi \rangle$. Assume that $\pi = f(v_A, v_s)$. Since $\pi_0 = f(v_0^A, v_0^A)$, this holds for $j = 0$. Assume that in $p_T$, the transducer $T$ transits in the step $j$ from $\langle s, v_s \rangle$ to $\langle s', v_s' \rangle$, while reading $\langle i, o \rangle$ and outputting $\langle o, o \rangle$. Note that the respective letter of the computation $w_T$ is $s' = \langle (i, o), g^t_r, r_s, a_s \rangle$, where $g^t_r = \langle v_s, i \rangle$ and it holds that $\langle s', v_s', o, r_s \rangle = \tau(s, i, g^t_r)$. Let $\langle q', v_A' \rangle$ be a $(\langle i, o \rangle, v_A')$-successor of $\langle q, v_A \rangle$ as appears in $\rho_{A'}$. In the full version [23], we prove that the pair $\langle q', \pi' \rangle$ is a $\sigma'$-successor of $\langle q, \pi \rangle$ in $A'$, where $\pi' = f(v_A', v_s')$. By repeatedly applying the above claim, we can start from $\langle q_0, \pi_0 \rangle$ and, for all $j \geq 0$, get the successor $\langle q_{j+1}, \pi_{j+1} \rangle$ of $\langle q_j, \pi_j \rangle$, obtaining the sought $\rho_{A'}$. Also, by the definition of $\alpha'$, the fact $\rho_{A'}$ is rejecting implies that so is $\rho_{A'}$, and so we are done.

Assume now that $T' \not\models A'$. We prove that $T \not\models A$. Since $T' \not\models A'$, there is an input sequence $w_T'$ that induces the run $p_T = s_0, s_1, \ldots$ and the computation $w_T'$ of $T'$ such that $w_T'$ generates a rejecting run $\rho_{A'} = \langle q_0, \pi_0 \rangle$ in $A'$. Given $w_T'$, and hence $p_T$, we construct a computation $w_T$ of $T$ that induces a rejecting run $\rho_A$ in $A$. The run $p_T$ starts in $\langle s_0, v_0^A \rangle$, and the run $\rho_A$ starts in $\langle q_0, \pi_0 \rangle$. Suppose that in some step $j \geq 0$, the run $p_T$ reaches a state $s$, the run $\rho_{A'}$ reaches a state $\langle q, \pi \rangle$, the run $p_T$ reaches a configuration $\langle s, v_s \rangle$, and the run $\rho_A$ reaches a configuration $\langle q, v_A \rangle$. Assume that $\pi = f(v_s, v_A)$. This holds for $j = 0$. Assume that $T'$ transits into $s'$ when reading $\langle i, o \rangle$ and outputting $\langle o, o \rangle$, and that $A'$ transits into $\langle q', \pi' \rangle$ when reading $\langle (i, o), g^t_r, r_s, a_s \rangle$. Then, as we prove in the full
version [23], there exist $i \in D$ such that the transducer $T$ transits into $\langle s', v'_0 \rangle$ on reading $\langle i, t \rangle$, the automaton $A$ transits into $\langle q'_0, v'_1 \rangle$ on reading $\langle (i, o), i, o \rangle$, where $o = v'_0(r_s)$, and $f(v'_0 \cup v'_1) = \pi'$. Applying the above claim in the initial step, when $j = 0$, we construct the configuration $\langle s_1, v'_1 \rangle$ of $\rho_T$, the configuration $\langle q_1, v'_1 \rangle$ of $\rho_A$, and the first letter $\langle (i, o), i, o \rangle$ of $w_T$. Note that the claim preconditions hold, in particular, $f(v'_1 \cup v'_1) = \pi_1$, so we can apply it again. By an iterative application, we construct the sought computation $w_T$ and the rejecting run $\rho_A$ on $w_T$.

We can now analyze the complexity of our synthesis algorithm. Recall that the input to the problem is a reg-UPW $A$ and an integer $k_s \geq 0$, and the output is a $k_s$-register transducer that realizes $A$, or an answer that no such transducer exists. Theorem 7 reduces the problem for $A$ with $n$ states, index $c$, and $k_A$ registers, to the synthesis problem of a (register-less) UPW $A'$ with $n(k_A + k_s)^{k_s + k_A}$ states and index $c$. Indeed, the state space of $A'$ is the product of that of $A$ with the set of possible partitions of the registers of $A$ and these of the generated transducer, and the number of such partitions is bounded by $(k_A + k_s)^{k_s + k_A}$. Note that $A'$ is parameterized by both $k_s$ and $\pi_0$. While $k_s$ is fixed, $\pi_0$ depends on the initial partition of $R_A$. Thus, we may need to repeat the reduction $|\Pi_s| \leq k_s^{c_s}$ times, where $\Pi_s$ is the set of system partitions. By [29, 32] a UPW with $N$ states and index $c$ can be determinized to a DPW with $(Nc)^{O(Nc)}$ states and index $O(Nc)$. Then, the synthesis problem for DPW reduces linearly, up to a multiplicative factor in the sizes of the alphabets, to solving parity games, which can be done in time at most $O((n')^5)$, for a game with $n'$ vertices and index $c' < \log n'$ [7]. The alphabet of $A'$ is $\Sigma' = \Sigma \times \mathbb{B}^{R_A} \times R_A \times \mathbb{B}^{R_A}$. Let $m = |\Sigma|$. Then, $|\Sigma'| = m \cdot 2^{O(k_s)}$. Thus, the new factor in the complexity is $|\Sigma|$, which is typically much smaller than $N$. It follows that the synthesis problem for $A'$ can be solved in time $(Nmc)^{O(Nc)} = (c^m(k_A + k_s)^{k_s + k_A})^{O(c^m(k_A + k_s)^{k_s + k_A})}$. Thus, a naive analysis gives a complexity that is doubly-exponential in $k_A$ and $k_s$ and is exponential in $n$ and $c$. As we argue below, the analysis can be tightened to a one that is doubly-exponential only in $k_A$ and is exponential in $n$, $c$, and $k_s$. Essentially, this follows from the fact that while the partition-component in the state space of $A'$ behaves universally with respect to the registers in $R_A$, it is deterministic with respect to these in $R_s$. Consequently, when counting the number of states in the DPW obtained by determinizing $A'$, we can replace the number of all possible partitions of $R$ by the number of partitions of $R$ for a fixed partition of $R_s$. For more details, see [23].

**Theorem 8.** Register-bounded synthesis with $k_s$ system registers for reg-UPWs with $n$ states, finite alphabet of size $m$, index $c$, and $k_A$ registers, is solvable in time $(m^c(k_A + k_s))^{O(m^c(k_A + k_s)^{k_s + k_A})}$. Thus, it is polynomial in $m$, exponential in $n$, $c$, and $k_s$, and doubly-exponential in $k_A$.

We note that when the specification automaton $A$ is a reg-UCW, its abstraction $A'$ is a UCW. Since reg-UCWs can be expressed as reg-UPWs with $c = 2$, the obtained time complexity for the case where specifications are reg-UCWs is $(m^c(k_A + k_s))^{O(m^c(k_A + k_s)^{k_s + k_A})}$.

## 4 Synthesis with a Fixed Number of System and Environment Registers

In this section, we consider the system-bounded synthesis problem with respect to restricted environments. Such environments are expressible by a register transducer with a bounded number of registers. Clearly, restricting the environments makes more specifications realizable. As we shall see, however, the complexity of the synthesis problem increases. An important
Register-Bounded Synthesis

conceptual difference between the setting studied in Section 3 and the one here is that once we fix the number of registers of both the system and the environment, we also fix the number of data values that may participate in the interaction. Indeed, the only data outputs that the system and environment transducers may generate during the interaction are those stored in their registers in their initial valuations.

In order to define the bounded setting, we first have to define the interaction between system and environment transducers. Consider a system transducer $T_{sys} = \langle \Sigma_i, \Sigma_O, S_i, s_0^i, R_i, v_0^i, \tau_i \rangle$ and an environment transducer $T_{env} = \langle \Sigma_O, \Sigma_i, S_e, s_0^e, R_e, v_0^e, \tau_e \rangle$. Note that the outputs of the environments are the inputs of the system, and vice versa. We denote the computation that is the interaction between the two transducers by $T_{env}|T_{sys}$, indicating that the environment initiates the interaction and is the first transducer to move.

Recall that $\tau_e : S_e \times (\Sigma_O \times B^{R_e}) \rightarrow S_e \times B^{R_e} \times \Sigma_i \times R_i$. The $\Sigma_O$ and $B^{R_e}$ components of the transition depend on the output of the system, which are generated when the system moves between states. Likewise, $\tau_s : S_s \times (\Sigma_i \times B^{R_i}) \rightarrow S_s \times B^{R_i} \times \Sigma_O \times R_s$, with the $\Sigma_i$ and $B^{R_i}$ components depending on the output of the environment. Recall that we assume that the environment moves first. Accordingly, for the first step of the interaction we assume that the $\Sigma_O$ and $B^{R_e}$ components are induced by the pair $\langle \varnothing, v_0(r_0) \rangle$, for some designated register $r_0 \in R_e$.

Formally, $T_{env}|T_{sys} = (\langle i_0, o_0, i_0, o_0 \rangle(\langle i_1, o_1, i_1, o_1 \rangle ... \in (\Sigma_i \times \Sigma_O \times D \times D)^0$ is such that there are runs $\rho_s = (s_0^0, v_0^0)\langle s_1^0, v_1^0 \rangle \langle s_2^0, v_2^0 \rangle ... \in (S_s \times D^{R_s})^0$ of $T_{env}$ and $\rho_e = (s_0^e, v_0^e)\langle s_1^e, v_1^e \rangle \langle s_2^e, v_2^e \rangle ... \in (S_e \times D^{R_e})^0$ of $T_{sys}$ such that the following hold: Let $\langle o_{-1}, o_{-1} \rangle = \langle \varnothing, v_0^e(r_0^e) \rangle$. Then, for every $j \geq 0$, the following hold:

- $\tau^e(s_j^e, o_{j-1}, v_j^e) \sim o_{j-1} = (s_{j+1}^e, a_j^e, i_j, r_j^e)$, $i_j = v_j^e(r_j^e)$, and $v_{j+1}^e = update(v_j^e, a_j^e, o_{j-1}^e)$.

That is, in each round in the interaction, including the first round, the environment moves first, the configuration $(s_{j+1}^e, v_{j+1}^e)$ is the $(o_{j-1}, o_{j-1})$-successor of $(s_j^e, v_j^e)$, and the transition taken in this move fixes $i_j$ and $o_j$.

- $\tau^s(s_j^s, i_j, v_j^s) \sim i_j = (s_{j+1}^s, a_j^s, o_j, r_j^s)$, $o_j = v_j^s(r_j^s)$, and $v_{j+1}^s = update(v_j^s, i_j, a_j^s)$. That is, the system respond by moving to the configuration $(s_{j+1}^s, v_{j+1}^s)$, which is the $(i_j, i_j)$-successor of $(s_j^s, v_j^s)$, and the transition taken in this move fixes $o_j$ and $a_j$.

The environment-system-bounded realizability problem is to decide, given a reg-UPW $A$ over $\Sigma_i \times \Sigma_O \times D \times D$, and numbers $k_s$ and $k_e$ of system and environment registers, respectively, whether there is a system transducer $T_{sys}$ with at most $k_s$ registers such that for all environment transducers $T_{env}$ with at most $k_e$ registers, we have that $T_{env}|T_{sys} \models A$.

The environment-system-bounded synthesis problem is to construct such a system transducer, if exists.

Let $A = (\Sigma, Q, q_0, R_A, v_0^A, \delta, \alpha)$. As in the construction in Section 3, we define a (registerless) UPW $A'$ that abstracts the registers of $A$ and maintains instead the equivalence relation between the registers. Here, however, the equivalence relation refers to the registers of $A$, of the system, and of the environment. Let $R_i, R_e$ denote the sets of system and environment registers, respectively. Let $R = R_i \cup R_e \cup R_A$. $\Pi$ be the set of equivalence relations over $R$, and $f : D^R \rightarrow \Pi$ is a map a register valuation to the partition it induces. We modify the function $update'$ from Section 3 to refer to registers directly, namely $update' : \Pi \times R \times B^{R} \rightarrow \Pi$ maps $\langle \pi, r, a \rangle$ to the partition resulting from moving the registers in $a$ into the equivalence class of $r$. Formally, $update'(\pi, r, a) = (S \setminus a : S \in \pi \cap C) \setminus \{\varnothing\} \cup \{C \cup a\}$, where $C \in \pi$ and $r \in C$.

The update function has properties similar to those stated in Lemma 5.

**Lemma 9.** For every valuation $v \in D^R$, register $r \in R$, and storing mask $a \subseteq R$, we have that $f(update(v, v(r), a)) = update'(f(v), r, a)$. 

> **Lemma 9.** For every valuation $v \in D^R$, register $r \in R$, and storing mask $a \subseteq R$, we have that $f(update(v, v(r), a)) = update'(f(v), r, a)$. 


Given a reg-UPW $A$, bounds $\mathcal{k}, \mathcal{k}_r \in \mathbb{N}$, and an initial partition $\pi_0 \in \mathcal{D}^R$, the $(\mathcal{k}, \mathcal{k}_r, \pi_0)$-abstraction of $A$ is the UPW $A' = (\Sigma', Q', q'_0, \delta', \alpha')$, defined as follows.

- $\Sigma' = \Sigma \times \mathcal{R} \times \mathbb{B}^R \times \mathbb{B}^R$.
- $Q' = (Q \times \Pi \times \mathcal{R}) \cup \{q'_0\}$. A state $(q, \pi, r_s) \in Q \times \Pi \times \mathcal{R}$ contains, in addition to the original state $q$ and partition $\pi$, the register $r_s$ whose value was output by the system transducer in the previous move.
- The initial state $q'_0 = \langle q_0, \pi_0, r'_0 \rangle$. It contains the environment register $r'_0$, because in the first move the environment transducer reads its own data value $v_0(r'_0)$.

Defining $\delta'$, we use two auxiliary partitions: First, $\pi^*$ corresponds to the register valuation after the environment transducer moves and updates its registers. Then, $\pi^{**}$ corresponds to the register valuations after the system transducer moves and updates its registers. Finally, the destination partition $\pi'$ corresponds to the register valuation after $A$ moves. For every state $(q, \pi, r) \in (Q \times \Pi \times \mathcal{R}) \cup \{\langle q_0, \pi_0, r'_0 \rangle\}$ and letter $(\sigma, r_s, g'_s, a^s) \in \Sigma'$, we have that $(q', \pi', r_s) \in \delta'(q, \pi, r, (\sigma, r_s, g'_s, a^s))$ if there exist $r_e \in \mathcal{R}_e$, $a'_s \in \mathbb{B}^R$, and $a^s_t \in \mathbb{B}^{R_A}$ satisfying the following.

- Let $\pi^* = update'(\pi, r, a^s_t)$. That is, the environment transducer updates its registers using the previous system value. (In the initial state, the environment transducer uses the value stored in its register $r'_0$.)
- Let $C \in \pi^*$ be the set that contains $r_e$. We require that $(C \cap \mathcal{R}_e) = g'_e$.
- Let $\pi^{**} = update'(\pi^*, r_e, a^s_t)$. That is, the system transducer updates its registers using the value stored currently in the register that the environment outputs.
- The automaton $A$ transits and updates its registers using the values in the registers of the environment and system transducers. Hence, the input guard $g'_s$ is $A$-chosen by $(\pi^{**}, r_e)$, while the output guard $g^s_t$ is $A$-chosen by $(\pi^{**}, r_s)$. Thus, we require that $(q', a^s_t) \in \delta(q, (\sigma, g'_s, g^s_t))$ and $\pi' = update'(\pi^{**}, r_e, a^s_t)$.
- The acceptance condition of $A'$ is induced from the one of $A$. Thus, for every state $(q, \pi, r) \in Q'$, we have that $\alpha'(\langle q, \pi, r \rangle) = \alpha(q)$.

Recall that the abstraction of $A$ is parameterized by both the number of registers that the system transducer may have as well as an initial partition for the registers of the system, the environment, and the automaton. Let $v_A \in \mathcal{D}^{R_A}$ be a valuation of the automaton registers, and $\pi_s$ a partition of $\mathcal{R}_s$. A partition $\pi \in \Pi$ is consistent with $v_A$ and $\pi_s$ if there are register valuations $v_e \in \mathcal{D}^{R_e}$ and $v_c \in \mathcal{D}^{R_c}$ s.t. $\pi_s = f(v_e)$ and $\pi = f(v_A \cup v_e \cup v_c)$. Thus, automata registers are related according to $v^A$, system registers are related according to $\pi_s$, and environment registers are not related in any special way.

**Theorem 10.** Consider a reg-UPW $A$ with $\Sigma = \Sigma_I \times \Sigma_O$, set of registers $R_A$, and an initial valuation $v^A_0$. Then, $A$ is realizable by a $\mathcal{k}_s$-register $\Sigma_I/\Sigma_O$-transducer with a set of registers $R_s$ with respect to environments that are $\mathcal{k}_r$-register $\Sigma_O/\Sigma_I$-transducers iff there is a partition $\pi_s$ of $\mathcal{R}_s$ and a $(\Sigma_I \times \mathcal{D}^{R_s})/(\Sigma_O \times \mathcal{R}_s \times \mathbb{B}^{R_c})$-transducer $T'$ such that for every partition $\pi_0$ of $R$ that is consistent with $v^A_0$ and $\pi_s$, the transducer $T'$ realizes the $(\mathcal{k}, \mathcal{k}_r, \pi_0)$-abstraction of $A$.

**Proof sketch.** The theorem follows from the following claim, which we prove in [23]. Fix a system $\mathcal{k}_s$-register transducer $T_{sys}$ with an initial valuation $v^A_0$, and fix an environment initial valuation $v^e_0$. Let $A'$ be the $(\mathcal{k}, \mathcal{k}_r, \pi_0)$-abstraction of $A$ with $\pi_0 = f(v^A_0 \cup v^e_0 \cup v^c_0)$. Let $T'_{sys}$ be the register-less transducer corresponding to $T_{sys}$. Then, we have that $T_{sys} = A'$ iff for every environment transducer $T_{env}$ with the initial valuation $v^e_0$, it holds that $T_{env} \parallel T_{sys} = A$. ▲
We now analyze the complexity of the environment-system-bounded synthesis problem. Using Theorem 10, we can reduce the synthesis problem for $k_s$ system and $k_c$ environment registers, reg-UPW $A$ with $n$ states, index $c$, and $k_A$ registers, to the synthesis problem of a (register-less) UPW $A'$ with $O(nk^k)$ states and index $c$, where $k = k_s + k_c + k_A$. Recall that the reduction does not create a single instance of the register-less synthesis problem, and instead requires to find a system partition $\pi_s$ such that the $(k_s, k_c, \pi_0)$-abstractions of $A$, for every $\pi_0$ consistent with $\psi_0^A$ and $\pi_s$, are realized by a single transducer. There can be no more than $k_s^k$ system partitions, and we are going to enumerate them one by one. Now, once a system partition $\pi_s$ is fixed, we can create a single UPW that represents the intersection of the abstraction UPWs for each $\pi_0$ consistent with $\pi_s$ and $\psi_0^A$. To this end, we create one initial state per $\pi_0$, while the rest of the definition stays the same. The number of initial states is bounded by $(k_s + k_c + k_A)^k$. Let us call this automaton $A'$. By the same naive analysis as in the system-bounded case, the synthesis problem for $A'$ can be solved in time $(Nmc)^{(N^c)} = (cnm^k)^{(c^{nk^k})}$, where $m = |\Sigma|$ is the size of the finite alphabet of $A$. In order to account for enumeration of system partitions, we multiply it by $k_s^k$, but this does not affect the asymptotic complexity. Thus, the environment-system-bounded synthesis problem is doubly-exponential in $k_A$, $k_s$, and $k_c$, and is exponential in $n$ and $c$.

As in the case of system-bounded synthesis, we can use the fact that the system-partition component in the state space of $A'$ is deterministic with respect to the registers in $R_s$, and behaves universally only with respect to the registers in $R_A$ and $R_c$. The universal behavior with respect to $R_c$ follows from the fact that a system transducer plays against all possible environment transducers. Accordingly, we can tighten the complexity as follows.

**Theorem 11.** Environment-system-bounded synthesis with $k_s$ system and $k_c$ environment registers for reg-UPWs with $n$ states, finite alphabet of size $m$, index $c$, and $k_A$ registers is solvable in time $(cnm(k_s + k_c + k_A))^{O(cn(k_s + k_c + k_A)^{(k_s + k_c + k_A + 1)})}$. Thus, it is polynomial in $m$, exponential in $c$, $n$, and $k_s$, and doubly-exponential in $k_A$ and $k_c$.

---

**References**

A. Khalimov and O. Kupferman 25:15


Register-Bounded Synthesis


