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 $T \parallel T'$, 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.

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, 31]. 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].

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 [38, 21, 27], software with integer parameters [5], communication protocols with message parameters [11], datalog systems with infinite data domain [4, 37], 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 [34], pebbles [29, 35], or variables [19, 20], or handle the infinite alphabets by attributing it by labels from an auxiliary finite alphabet [3, 2].

A register automaton [34] 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, [22] forces the content of the registers to be different, [29] adds alternation and two-wayness, [23] allows the registers to change their content nondeterministically during the run, and [36] 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 [29], 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 [16], 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
A. Khalimov and O. Kupferman 21:3

with a bounded number of registers. As has been the case with bounded synthesis in the finite-state setting [32, 13, 17, 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.

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 [32, 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 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 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 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 the environment are bounded ([24] only bounds the system), and we study the theoretical

---

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, [18] describes a CEGAR-based synthesis algorithm that approaches the general undecidable synthesis problem by iteratively refining under-approximating systems of bounded sizes.
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 \( (cn(k_s + k_e + k_A))^{O(cn(k_s + k_e + k_A)(k_s + k_e + 2))} \). Thus, it is polynomial in \( m \), exponential in \( c \), \( n \), and \( k_s \), and doubly-exponential only in \( k_A \) and \( k_e \). Then, in Section 5 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]. Finally, in Section 6, we 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 \( (\sigma_0, i_0, o_0)(\sigma_1, i_1, o_1)... \in (\Sigma \times D \times D)^\omega \). Register automata specify languages of such words. Let \( B = \{true, false\} \). A k-register word automaton is a tuple \( A = (\Sigma, Q, q_0, R, o, \delta, \alpha) \), 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, \( o \in D^R \) is an initial register valuation, \( \delta : Q \times (\Sigma \times B^R \times B^R) \rightarrow 2^{Q \times B^R} \) 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 \( (\sigma, i, o) \in \Sigma \times D \times D \), 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 \( \Sigma \times B^R \times B^R \), where the two guards in \( B^R \) compare the values stored in the registers with \( i \) and \( o \). Then, \( \delta \) directs \( A \) into a set of pairs in \( Q \times B^R \), each describing a successor state and a storing mask, indicating which registers are going to store \( i \).

A configuration of \( A \) is a pair \( (q, v) \in Q \times D^R \), describing the state that \( A \) visits and the content of its registers. A run of \( A \) starts in the configuration \( (q_0, v_0) \), 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 \( g \in B^R \) that indicates the agreement of \( v \) with \( d \). Thus, for every \( r \in R \),
we have $g(r) = \text{true}$ iff $v(r) = \delta$. The function $\text{update} : \mathcal{D}^R \times \mathcal{D} \times \mathbb{B}^R \to \mathcal{D}^R$ maps a valuation $v \in \mathcal{D}^R$, a value $\delta \in \mathcal{D}$, 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 $\text{update}(v, \delta, a)(r) = \delta$ if $a(r) = \text{true}$ and is $v(r)$ otherwise. Note that it need not be the case that $\text{update}(v, \delta, a)(r) = \delta$ regardless of $a(r)$.

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

Now, a run of $A$ on a word $w = \langle \sigma_0, i_0, o_0 \rangle \langle \sigma_1, i_1, o_1 \rangle \ldots \in (\Sigma \times \mathcal{D} \times \mathcal{D})^\omega$ is an infinite sequence $\langle \sigma_0, v_0 \rangle \langle \sigma_1, v_1 \rangle \ldots \in (\mathcal{D} \times \mathcal{D})^\omega$ of configurations such that for every $j \geq 0$, we have that $\langle \sigma_{j+1}, v_{j+1} \rangle$ is a $\langle \sigma_j, i_j, o_j \rangle$-successor of $\langle \sigma_j, v_j \rangle$. 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 $\langle \sigma_j, v_j \rangle$ 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, $\alpha : Q \to \{0, \ldots, c - 1\}$, 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, $\rho = \langle \sigma_0, v_0 \rangle \langle \sigma_1, v_1 \rangle \ldots$ is accepting if $\max\{j \in \{0, \ldots, c - 1\} : \alpha(\sigma_j) = j\}$ 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 $\langle q, v \rangle$ with $\alpha(\sigma) = 1$ 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 specific case of a reg-UPW with no registers. In particular, it has no initial valuation and its transition function is of the form $\delta : Q \times \Sigma \to 2^Q$.

**Example 1.** The reg-UCW $A$ appearing in Figure 1 specifies an arbiter with a single output signal $\text{ack}$ (that is, $\Sigma_I$ is a singleton, and we ignore it, and $\Sigma_O = 2^{\{\text{ack}\}}$) that gets in each moment in time an input data value $i$, and outputs either $\text{ack}$ or $\neg \text{ack}$ 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 $\text{ack}$. 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 $\Sigma \times \mathcal{D} \times \mathcal{D}$ by triples in $\{\text{ack}, \neg \text{ack}\} \times \{0, 1\} \times \{0, 1\}$, possibly replacing some of the parameters by $\_$, indicating that both values of this parameter apply. We continue to describe the transition function. First, $\delta(q_0, \_ , \_ , \_ ) = \{q_0, 0\}$. That is, if the input data value agrees with the one stored in $r_1$, we only loop in $q_0$. Then, $\delta(q_0, \_ , 0, \_ ) = \{q_0, 1\}$. 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 $\delta(q_1, \{\text{ack}, \_ , 0\}) = \emptyset$, thus the copy sent to $q_1$ fulfills its mission when it reads an $\text{ack}$ 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, $\delta(q_1, \{\neg \text{ack}, \_ , 0\}) = \{q_1, 0\}$. The parity acceptance condition $\alpha = \{q_0 \rightarrow 0, q_1 \rightarrow 1\}$ 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.
\[ i = r_1 \]
\[ i \neq r_1/\text{store}_1 \]
\[ \neg \text{ack} \lor o \neq r_1 \]

![Figure 1](image-url) 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 \( \text{store}_1 \) 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.

### 2.2 Register Transducers

Register transducers model systems with inputs in \( \Sigma_I \times \Delta \) and outputs in \( \Sigma_O \times \Delta \). Every such system implements a strategy \( (\Sigma_I \times \Delta)^+ \rightarrow \Sigma_O \times \Delta \), describing the output it generates after reading a sequence of inputs. A register transducer is a tuple \( T = (\Sigma_I, \Sigma_O, S, s_0, R, \nu_0, \tau) \), 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, \( \nu_0 \in \Delta^R \) is an initial register valuation, and \( \tau : S \times (\Sigma_I \times \Delta)^+ \rightarrow S \times \Delta^R \times \Sigma_O \times R \) is a transition function. Intuitively, when \( T \) is in state \( s \) and reads a letter \( (i, o) \in \Sigma_I \times \Delta \), 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 \( S \times \Delta^R \), and successive configurations are defined in a way similar to the one defined for automata, except that \( T \) is deterministic: given a configuration \( \langle s, v \rangle \in S \times \Delta^R \) and an input \( (i, o) \in \Sigma_I \times \Delta \), let \( \tau(s, (i, v) \sim i) = \langle s', a, o, r \rangle \). Then, the \( (i, i) \)-successor of \( \langle s, v \rangle \) is \( \langle s', \text{update}(v, i, a) \rangle \).

Given an input word \( w = \langle i_0, o_0 \rangle \langle i_1, o_1 \rangle \ldots \in (\Sigma_I \times \Delta)^{\omega} \), the run of \( T \) on \( w \) is the sequence \( \langle s_0, v_0 \rangle \langle s_1, v_1 \rangle \ldots \in (S \times \Delta^R)^{\omega} \), where for all \( j \geq 0 \), we have that \( \langle s_j, v_j \rangle \sim i_j \) is the \( (i, i) \)-successor of \( \langle s_j, v_j \rangle \). For every \( j \geq 0 \), let \( \tau(s_j, i_j, v_j \sim i_j) = \langle s_{j+1}, a_j, o_j, r_j \rangle \). Then, the computation of \( T \) on \( w \) is the sequence \( \langle i_0, o_0 \rangle \langle i_1, o_1 \rangle \langle i_2, o_2 \rangle \ldots \in ((\Sigma_I \times \Sigma_O) \times \Delta^R \times \Delta)^{\omega} \) such that for every \( j \geq 0 \), we have that \( o_j = \text{update}(v_j, i_j, o_j)(r_j) \). Thus, the transducer moves from \( s_j \) to \( s_{j+1} \), stores \( i_j \) in registers that are positive in \( o_j \), 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 \( w \in (\Sigma_I \times \Delta)^{\omega} \), 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 \( \Sigma_O = \{\text{ack}\} \), 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 $ack$. 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 $ack$, 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 $acks$, while our transducer does not.

\[ i = r_1/\langle \neg ack, r_1 \rangle \]
\[ i \neq r_1/\langle \neg ack, r_1, store_1 \rangle \]
\[ i = r_1/\langle ack, r_1 \rangle \]
\[ i \neq r_1/\langle ack, r_1, store_2 \rangle \]
\[ i \neq r_2/\langle ack, r_2, store_1 \rangle \]
\[ i = r_2/\langle ack, r_2 \rangle \]
\[ i \neq r_2/\langle \neg ack, r_2, store_2 \rangle \]
\[ i = r_2/\langle \neg ack, r_2 \rangle \]

Figure 2 A register transducer that realizes the reg-UCW $A$ from Example 1. The edge labeling for $\Sigma_O$ and the guards is symbolic, and is similar to that in Figure 1.

### 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.

➤ **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.*

### 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 $k_A$ of registers, whether there is a transducer with at most $k_A$ registers all whose computations are accepted by $A$. The system-bounded synthesis problem is to construct such a transducer, if exists.

Let $A = (\Sigma, Q, q_0, R_A, \delta_A, \alpha)$, 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 $v_A \in D^{R_A}$ and $v_s \in D^{R_s}$, let $v = v_A \cup v_s$ be the valuation in $D^R$ obtained by taking their union. Likewise, for a valuation $v \in D^R$, 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 $f : D^R \rightarrow \Pi$ map a register valuation $v \in D^R$ to the partition $\pi \in \Pi$, where for every two registers $r, r' \in R$, we have that $\pi(r, r')$ iff $v(r) = v(r')$.

Recall that we describe guards and storing masks on a set $R$ of registers by Boolean functions in $B^R$. Each assignment $g \in B^R$ 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 $S \in \pi \cup \{\varnothing\}$ such that $S \cap R_s = g_s$. We then say that $(\pi, g_s)$ chooses $S$. Note that for $g_s = \varnothing$, 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 $(\pi, \varnothing)$ chooses $\{1\}, \{2, 3\}$, or $\varnothing$. For a set $S_A \subseteq R_A$, we say that $(\pi, g_s)$ $\pi$-chooses $S_A$ if there is a set $S \in \pi \cup \{\varnothing\}$ such that $(\pi, g_s)$ chooses $S$ and $S_A = S \cap R_A$. Thus, $(\pi, g_s)$ $\pi$-chooses $S_A$ if $(\pi, g_s)$ chooses a set whose $R_A$ registers are in $S_A$. Continuing the previous example, $(\pi, \{s_5\})$ $\pi$-chooses $\{s_4\}$, the pair $(\pi, \{s_6\})$ $\pi$-chooses $\varnothing$, and $(\pi, \varnothing)$ $\pi$-chooses $\{1\}, \{2, 3\}$, or $\varnothing$. Finally, for a register $r \in R$, the pair $(\pi, r)$ $\pi$-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 $(\pi, s_4)$ and $(\pi, s_5)$ both $\pi$-choose $\{s_4\}$, and the pair $(\pi, s_6)$ $\pi$-chooses $\varnothing$.

The following lemma follows immediately from the definitions.

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

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

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

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

Recall the function $update : D^R \times D \times B^R \rightarrow D^R$, where $update(v, d, a)$ is obtained from $v$ by storing $d$ in the registers in $a$. We now define a function $update' : \Pi \times B^R \times B^R \rightarrow \Pi$, which adjusts the update function to the abstraction of valuations by partitions. Intuitively, for a partition $\pi \in \Pi$, a guard $g \in (\pi \cup \{\varnothing\})$, 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, $update'(\pi, \varnothing, a) = \{S \setminus a : S \in \pi \setminus \{\varnothing\} \cup \{a\}$.

> **Lemma 5.** For every valuation $v \in D^R$, value $i \in D$, 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 $k_s$, the abstraction is parameterized by a partition $\pi_0$ of the system and automaton registers. Given $k_s$ and $\pi_0$, the $(k_s, \pi_0)$-abstraction of $A$ is the UPW $A' = (\Sigma', Q', \delta', \alpha')$ 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 $f(v_s \cup v_A) = \pi$.

- $\Sigma' = \Sigma \times B^{R_s} \times R_s \times B^{R_s}$. Recall that in $A$, the transition function is $\delta : Q \times (\Sigma \times B^{R_s} \times B^{R_s}) \to 2^{Q \times B^{R_s}}$, and when $A$ is in configuration $\langle q, v_A \rangle$ and reads a letter $\langle \sigma, i, o \rangle \in \Sigma \times D \times D$, it proceeds according to $\langle \sigma, q_i^A, g_i^A \rangle \in \Sigma \times B^{R_s} \times B^{R_s}$, where $g_i^A$ is $v_A \sim i$ and $g_i^A$ is $v_A \sim o$. Also, each successor state $q'$ is paired with a storing mask $a_i^A \in B^{R_s}$, which induces a successor configuration $\langle q', \text{update}(v, i, a_i^A) \rangle$. Intuitively, each letter $\langle \sigma, q_i^A, r_s, a_i^A \rangle \in \Sigma'$, together with the current partition, induces choices for $\langle \sigma, q_i^A, g_i^A, a_i^A \rangle \in \Sigma \times B^{R_s} \times B^{R_s} \times B^{R_s}$ which determine the transitions in $A$ that the abstraction follows.

- For every state $\langle q, \pi \rangle \in Q'$ and letter $\langle \sigma, q_i^A, r_s, a_i^A \rangle \in \Sigma'$, we have that $\langle q', \pi' \rangle \in \delta'((q, \pi), (\sigma, q_i^A, r_s, a_i^A))$ if there exist $g_i^A, g_i^A, a_i^A \in B^{R_s}$ such that the following conditions hold.
  - $g_i^A$ is $A$-chosen by $\langle \pi, g_i^A \rangle$. Let $g_i = g_i^A \cup g_i^A$. Note that $g_i \in (\pi \cup \emptyset)$.
  - 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 $\pi^\ast$. Let $\pi^\ast$ be the partition after updating the system registers in $\pi$ according to the guard $g_i^A$ and the storing mask $a_i^A$. Thus, $\pi^\ast = \text{update}'(\pi, g_i^A)$. $g_i^A$ is $A$-chosen by $\langle \pi^\ast, r_s \rangle$. Note that since the set chosen by $\langle \pi^\ast, r_s \rangle$ is not empty, $v_s$ is unique.
  - $\langle q', a_i^A \rangle \in \delta(q, \langle \sigma, g_i^A, g_i^A \rangle)$.
  - We can now complete updating the partition. The partition $\pi'$ is the result of updating the registers of $A$ in $\pi^\ast$ according to the guard $g_i^A$ and the storing mask $a_i^A$. Let $g_i' = g_i \cup a_i^A$ be the updated guard after system storing. Then $\pi' = \text{update}'(\pi^\ast, g_i', a_i^A)$.
  - 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 $v_A \in D^{R_s}$ be a valuation of the automaton registers. A partition $\pi \in \Pi$ is consistent with $v_A$ if there is a register valuation $v_s \in D^{R_s}$ 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 $D = \mathbb{N}$, $R_A = \{\#, \$, , \#, \$, \#, \$, \#, \$\}$, and $R_s = \{\#5, \#6, \#7\}$. Then the partition $\pi = \{\{\#, \$, \#, \$\}, \{\#, \$, \$, \$\}, \{\#, \$, \$\}\}$ is consistent with the valuation $v_A \in D^{R_s}$ for which $v_A(\#1) = v_A(\#4) = 9, v_A(\#2) = 2$, and $v_A(\#3) = 13$. Indeed, taking $v_s \in D^{R_s}$ with $v_s(\#5) = 9, v_s(\#6) = 2$, and $v_s(\#7) = 14$ results in $\pi = f(v_A \cup v_s)$. Note that different valuations $v_s \in D^{R_s}$ may witness the consistency of $\pi$ with $v_A$. In our example, all these with $v_s(\#5) = 9, v_s(\#6) = 2$, and $v_s(\#7) \not\in \{2, 9, 13\}$. Also, several different partitions may be consistent with a given valuation $v_A \in D^{R_s}$. In our example, all these in which register $\#1$ and $\#4$ are in the same set, different from the (different) sets of $\#2$ and $\#3$. 
We can now state our main theorem, relating the realizability of A with realizability of its abstraction. Consider a $k_A$-register $\Sigma_I/\Sigma_O$-transducer $T = (\Sigma_I, \Sigma_O, S, s_0, R, v_0, T)$. 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 $v_0 \in \mathcal{D}^{R_s}$ 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 $k_A$-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 $v_A^0$. Then, $A$ is realizable by a $k_A$-register $\Sigma_I/\Sigma_O$-transducer with a set of registers $R_k$, consistent with $v_A^0$, such that the $(k_A, \pi_0)$-abstraction of $A$ is realizable by a $(\Sigma_I \times \mathbb{B}^{R_k})/(\Sigma_O \times R_k \times \mathbb{B}^{R_k})$-transducer. In particular, a transducer that realizes the $(k_A, \pi_0)$-abstraction of $A$ corresponds to a $k_A$-register transducer that realizes $A$.

**Proof.** Let $A = (\Sigma, Q, q_0, R_A, v_A^0, \delta, \alpha)$ and let $A'$ be its $(k_A, \pi_0)$-abstraction, such that $\pi_0$ is a partition of $R$ consistent with $v_A^0$. We prove that for every valuation $v_0^A \in \mathcal{D}^{R_k}$, satisfying $f(v_0^A \cup v_0^A) = \pi_0$, $k_A$-register $\Sigma_I/\Sigma_O$-transducer $T$ initialized with $v_0^A$, and registerless $(\Sigma_I \times \mathbb{B}^{R_k})/(\mathbb{B}^{R_k} \times \Sigma_O \times R_k)$-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 $w_T = \langle i_0, i_0 \rangle \langle i_1, i_1 \rangle \ldots$, a run $\rho_T = \langle s_0, v_0^A \rangle \langle s_1, v_1^A \rangle \ldots$ of $T$ on $w_T$, a computation $w_T = \langle i_{0,0}, o_0 \rangle \langle i_{0,1}, o_1 \rangle \langle i_{1,1}, o_1 \rangle \ldots$ that $T$ generates when it follows $\rho_T$, and a rejecting run $\rho_A = \langle q_0, v_0^A \rangle \langle q_1, v_1^A \rangle \ldots$ 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 $\rho_T$ induce an input sequence $w_{T'}$, to $T'$ such that $A'$ rejects the computation of $T'$ on $w_{T'}$. We define $w_{T'} = (i_0, v_0^A \sim i_0) \langle i_1, v_1^A \sim i_1 \rangle \ldots$. The word $w_{T'}$ uniquely defines the computation $w_{T'}$ and the run $\rho_{T'} = s_0 \circ \ldots$, 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 $\rho_T$ reaches the configuration $\langle s, v_s \rangle$, and the run $\rho_A$ reaches the state $\langle q, \pi \rangle$. Assume that $\pi = f(v_s \cup v_s)$. Since $\pi_0 = f(v_0^A \cup v_0^A)$, this holds for $j = 0$. Assume that in $\rho_T$, the transducer $T$ transit 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 $\pi' = \langle (i, o), r_s, a_s \rangle$, where $g_s = (v_s \sim i)$ and it holds that $\langle s', a_s, o, r_s \rangle = \tau(s, i, g_s)$. Let $\langle q', v_A' \rangle$ be a $(\langle i, o \rangle, o)$-successor of $\langle q, v_A \rangle$ as appears in $\rho_{A'}$. We argue that $\langle \langle q, \pi \rangle \rangle$ is a $\alpha'$-successor of $\langle q, \pi \rangle$ in $A'$, where $\alpha' = f(v_A' \cup v_A')$. By repeatedly applying (4), 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 run $\rho_{A'}$. Also, by the definition of $\alpha'$, the fact that $\rho_{A'}$ is rejecting implies that so is $\rho_{A'}$. It is left to prove the claim (4). By the definition of the $A'$ transitions, there exist $g_A = (v_A \sim i)$, the guard $g_A = (v_A \sim v_A' \cup v_A')$, and $\langle q', a_A \rangle = \delta(q, \langle i, o \rangle, g_A, g_A))$. We need to show the following:

- $g_A$ is $A$-chosen by $g_A$. This follows from Lemma 4(a), for $g_A = (v_A \sim i)$ and $g_A = (v_A \sim i)$.
- $g_A$ is $A$-chosen by $\langle \pi', r_s \rangle$, where $\pi' = \text{update}(\langle \pi, r_s \rangle, a_s)$. Note that $f(v_A' \cup v_A') = \pi'$ by Lemma 5, since $g_s = (v_s \sim i)$ and $f(v) = \pi$. The statement then follows from Lemma 4(c).
- The satisfaction of $\delta$ is immediate because of the $A$ transition.
- $\pi' = f(v_A' \cup v_A')$. By the abstraction definition, $\pi' = \text{update}(\pi', g_A, a_A)$, where $g_A = (g_A \cup a_A)$. Note that $v_A' \cup v_A' \cup a_A = \text{update}(v_A' \cup v_A', i, a_A)$. The item then follows from Lemma 5, since $g_A = (v_A' \cup v_A' \sim i)$. 


This concludes the proof of the claim ($\dagger$), and so $T \not\models A$ implies that $T' \not\models A'$.

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 $\rho_T = s_0 s_1 \ldots$ and the computation $w_T$ of $T'$ such that $w_T$ generates a rejecting run $\rho_{A'} = (q_0, \pi_0)(q_1, \pi_1) \ldots$ in $A'$. Given $w_T$ (and hence $\rho_T$) and $\rho_{A'}$, we construct a computation $w_T$ of $T$ that induces a rejecting run $\rho_{A}$ in $A$. The run $\rho_T$ starts in $\langle s_0, v_0 \rangle$, and the run $\rho_{A'}$ starts in $\langle q_0, v_0' \rangle$. Suppose that in some step $j \geq 0$, the run $\rho_T$ reaches a state $s$, the run $\rho_{A'}$ reaches a state $\langle q, \pi \rangle$, the run $\rho_T$ reaches a configuration $(s, v)$, and the run $\rho_{A'}$ reaches a configuration $\langle q, v_A \rangle$. Assume that $\pi = f(v_A \cup v_A)$. This holds for $j = 0$. Assume that $T'$ transits into $s'$ when reading $\langle i, g_i' \rangle$ and outputting $(a_s, o, r_s)$, and that $A'$ transits into $\langle q', \pi' \rangle$ when reading $\langle (i, o), g_i', r_s, a_s \rangle$. Then, ($\dagger$) there exist $i \in \mathcal{D}$ such that the transducer $T$ transits into $\langle s', v' \rangle$ on reading $\langle i, i \rangle$, the automaton $A$ transits into $\langle q', v'_A \rangle$ on reading $\langle (i, o), i, o \rangle$, where $v = v'_i(r_s)$, and $f(v'_i \cup v'_A) = \pi'$.

Applying ($\dagger$) 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 preconditions of ($\dagger$) hold, in particular, $f(v_1' \cup v_1') = \pi_1$, and we can apply it again. By an iterative application, we construct the sought computation $w_T$ and the rejecting run $\rho_{A'}$ on $w_T$. It is left to prove the claim $\dagger$. Let $v = v_i \cup v_A$. Because $\langle q, \pi \rangle$ transits into $\langle q', \pi' \rangle$, there exist $g_i', g_A' \subseteq A$ satisfying the conditions of the abstraction definition. In particular, $g_A'$ is $A$-chosen by $\langle v, g_A' \rangle$. Let $g_i' = g_i' \cup g_A'$; note that $g_i' \in (\pi \cup \{\varnothing\})$. By Lemma 4(b), there exists $i \in \mathcal{D}$ such that $v' \sim i = g_i$ (when $g_i = \varnothing$, the value $i$ is not unique). Define $v_i' = \text{update}(v_i, i, a_s)$. Because $v_i' \sim i = g_i'$, it is immediate that $T$ transits into $\langle v_i', v_A' \rangle$ on reading $\langle i, i \rangle$. Let $T$ transits into $\langle q', v'_A \rangle$ on reading $\langle (i, o), i, o \rangle$, we need to show the following:

- $\langle q', v'_A \rangle \in \delta(q, \langle (i, o), g_i', g_A' \rangle)$. This holds because $A'$ transits into $\langle q', \pi' \rangle$ on reading $\langle (i, o), g_i', g_A' \rangle$.
- $\langle v_A \sim i = g_A'. This follows from the way we picked $i$.
- $\langle v_A \sim o = g_A'. Recall that $v = v_i'(r_s)$. By definition of the abstraction, the guard $g_A'$ is $A$-chosen by $\langle \pi^*, r_s \rangle$, where $\pi^* = \text{update}(\pi, g_i, a_s)$. Furthermore, $f(v'_i \cup v_A) = \pi^*$ by Lemma 5, since $f(v) = \pi$ and $v'^* \sim i = g_i'$. Hence by Lemma 4(c), the pair $\langle \pi^*, r_s \rangle$ $A$-chooses $v_A \sim o$. By the abstraction definition, the pair $\langle \pi^*, r_s \rangle$ $A$-chooses $g_A'$. Since the choice is unique, they are equal.

Finally, we show that $f(v_i' \cup v_A') = \pi'$. By the abstraction definition, $\pi' = \text{update}(\pi^*, g_A', a_A)\rangle$, where $g_A' = g_A \cup a_A$. Note that $\langle (v_i' \cup v_A) \sim i = g_i'$ (i.e., the registers equal to $i$ that were equal to $i$ ($g_i$) union the updated ones ($a_s$)). The statement then follows from Lemma 5. This concludes the proof of the claim ($\dagger$), and so $T' \not\models A'$ implies that $T \not\models A$. \hfill \blacksquare

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 a fixed $n$ states, index $c$, and $k_s$ registers, to the synthesis problem of a (register-less) UPW $A'$ with $n(k_A + k_s)^{k_s + k^*}$ 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 $(k_A + k_s)^{k_s + k^*}$. 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 $R_s$. Thus, we may need to repeat the reduction $\Pi_s \leq k_s^*$ times, where $\Pi_s$ is the set of system partitions. By [30, 33] 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 $\Sigma' = \Sigma \times \mathbb{B}^{R_s} \times \mathbb{R} \times \mathbb{B}^{R_s}$. 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 $(Nm^c)^{O(Nc)} = (cmn(k_s+k_{\text{sys}})k_{\text{sys}})O(cn(k_s+k_{\text{sys}})^{2c_k})$. Thus, a naive analysis gives
a complexity that is doubly-exponential in $k_s$ and $k_{\text{sys}}$ 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
$k_s$ and is exponential in $n$, $c$, and $k_{\text{sys}}$. 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 $R_s$.

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 $(cmn(k_s+k_A))^{O(cn(k_s+k_A)^{2c_k})}$. Thus, it is polynomial in $m$, exponential in $n$, $c$, and $k_s$, and doubly-exponential in $k_A$.

Proof. By [33, 30], a UPW $A'$ with $N$ states and index $c$ can be determined to a DPW
with $(Nc)^{O(Nc)}$ states and index $O(Nc)$ 2. The state space of the DPW consists of history
trees. Each node in a history tree is a state of $A'$, their union is the set of states that $A'$
may visit, and the structure of the history tree maintains information required for detecting
the maximal rank that is visited infinitely often. Thus, the history-tree construction is an
augmentation of the subset construction, where on top of the subset of states, we maintain
additional information.

Recall that in the UPW $A'$, each state is a member of $Q \times \Pi$, where $\Pi$ is the set of
possible partitions of $R = R_A \cup R_s$. Also, for every letter in $\Sigma'$, namely $(q, g_s, r_s, a_s) \in
\Sigma \times \mathbb{B}^{R_s} \times \mathbb{R} \times \mathbb{B}^{R_s}$, the $(q, g_s, r_s, a_s)$-successors of a state $(q', \pi')$ are pairs $(q', \pi')$ in which
$\pi'$ is the partition obtained from $\pi$ by updating the system registers according to the
system assignment $a_s$ and updating the automaton registers according to the transition
of $A$ that induce the pair $(q', \pi')$. The update of the system registers is deterministic: all
the $(q, g_s, r_s, a_s)$-successors $(q', \pi')$ agree on the partition of the system registers in $\pi'$: this
partition is uniquely defined by the partition of the system registers in $\pi$, the system guard
g_s, and the storing mask $a_s$. In contrast, the automaton registers in $\pi'$ may be updated in
different ways, induced from the universal branches in $A$.

The deterministic behavior of the system registers enables us to tighten the bound on
the number of different history trees. Indeed, recall that each history tree is associated
with a subset of the states of $A'$. Rather than considering all subsets, we need to consider
only those in which the $\Pi$ components of all states agree on the partition of the system
registers. The number $x$ of such states is bounded by $n \cdot |\Pi|$, where $|\Pi|$ is the number of
partitions of $R$ for a fixed partition of $R_s$, and so $|\Pi| \leq (k_s + k_A)^{k_s}$. By [30, Claim 4.6]
the number of history trees is $(x)^{O(xc)}$. Therefore, the number of history trees, subject to a
fixed system partition, is $(cn(k_s+k_A))O(cn(k_s+k_A)^{k_s})$. To get the total number of history
trees, we multiply it by the number $|\Pi| \leq k_s^{k_s}$ of system partitions, which does not change the
asymptotic analysis.

In addition, by [30, p.15], the index of the DPW is bounded by $2(|\frac{2}{3}| + 1)x$, for $x$ as
above, which is $O(cn(k_s+k_A)^{k_s})$.

2 The paper [30] determinizes Streett automata, but we can convert a UPW of index $c$ into a Streett
automaton with $|\frac{2}{3}|$ acceptance pairs.
Given a DPW with \( n' = (cn(k_s + k_A))^O(cn k_s (k_s + k_A)^{k_e}) \) states and index \( c' = O(cn(k_s + k_A)^{k_e}) \), we construct the parity game of size \( mn' \) and index \( c' \). A winning strategy for the system player witnesses that \( A \) is realizable, and induces the sought system transducer. For \( c' < \log n' \), parity games can be solved in time \( O(n')^{\log n} \) [7]. This gives a \((cn(k_s + k_A))^O(cn k_s (k_s + k_A)^{k_e})\) time complexity for checking the realizability of \( A' \) for a single initial partition \( \pi_0 \). Repeating this for all possible \( |\Pi_s| \) such partitions does not change the asymptotic analysis.

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 \((mn(k_s + k_A))^O(cn k_s (k_s + k_A)^{k_e})\).

## 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 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 these 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 \( T_{sys} = \langle \Sigma_I, \Sigma_O, S_s, s_0^s, R_s, v_0^s, \tau_s \rangle \) 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 \( \tau_e : S_e \times (\Sigma_O \times \mathbb{R}^e) \rightarrow S_e \times \mathbb{R}^e \times \Sigma_I \times R_s \). The \( \Sigma_O \) and \( \mathbb{R}^e \) components of the transition depend on the output of the system, which are generated when the system moves between states. Likewise, \( \tau_s : S_s \times (\Sigma_I \times \mathbb{R}^e) \rightarrow S_s \times \mathbb{R}^s \times \Sigma_O \times R_e \), with the \( \Sigma_I \) and \( \mathbb{R}^e \) 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 \( \mathbb{R}^e \) components are induced by the pair \( \langle o, o_0(r_0) \rangle \), for some designated register \( r_0 \in R_e \).

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

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

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

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

That is, the system respond by moving to the configuration \( \langle s_{j+1}^e, v_{j+1}^e \rangle \), which is the \( \langle i_j, i_j \rangle \)-successor of \( \langle s_j^e, v_j^e \rangle \), and the transition taken in this move fixes \( o_j \) and \( o_j \).

The environment-system-bounded realizability problem is to decide, given a reg-UPW
Register-Bounded Synthesis

Let $A = (\Sigma, Q, q_0, R_A, v_0^i, \delta, \alpha)$. 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 $R_s$ and $R_e$ denote the sets of system and environment registers, respectively. Let $R = R_s \cup R_e \cup R_A$. Let $R$ be the set of equivalence relations over $R$, and $f : R \rightarrow \Pi$ 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 $(\pi, r, a)$ to the partition resulting from moving the registers in $a$ into the equivalence class of $r$. Formally, $update'((\pi, r, a)) = (S \setminus a : S \in \pi \setminus \{O\} \cup \{C \cup a\})$, where $C \in \pi$ and $r \in C$. The update function has properties similar to these stated in Lemma 5.

Lemma 9. For every valuation $v \in B^R$, 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 $k_s, k_e \in \mathbb{N}$, and an initial partition $\pi_0 \in \Pi$, the $(k_s, k_e, \pi_0)$-abstraction of $A$ is the UPW $A' = (\Sigma', Q', q_0', \delta', \alpha')$, defined as follows.

- $\Sigma' = \Sigma \times R_s \times B^{R_s} \times B^R$.
- $Q' = (Q \times \Pi \times R_s) \cup \{(q_0, \pi_0, r_0^s)\}$. A state $\langle q, \pi, r \rangle \in Q \times \Pi \times R_s$ 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 $q_0' = (q_0, \pi_0, r_0^s)$. It contains the environment register $r_0^s$, because in the first move the environment transducer reads its own data value $v_0^i(r_0^s)$.

- 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 $\langle q, \pi, r \rangle \in (Q \times \Pi \times R_s) \cup \{(q_0, \pi_0, r_0^s)\}$ and letter $\langle \pi, r_s, g_s^e, a_s^e \rangle \in \Sigma'$, we have that $\langle q', \pi', r_s \rangle \in \delta'(\langle q, \pi, r \rangle, \langle \pi, r_s, g_s^e, a_s^e \rangle)$ if there exist $r_e \in R_e, a_e^s \in B^{R_e}$, and $a_s^e \in B^{R_s}$ satisfying the following.

- Let $\pi^* = update'((\pi, r, a_s^e))$. 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 $r_0^s$.)

- Let $C \in \pi^*$ be the set that contains $r_e$. We require that $(C \cap R_s) = g_s^e$.

- Let $\pi^{**} = update'((\pi^*, r_s, a_s^e))$. 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 $g_s^A$ is $A$-chosen by $\langle \pi^{**}, r_s \rangle$, while the output guard $g_e^A$ is $A$-chosen by $\langle \pi^*, r_e \rangle$. Thus, we require that $\langle q', a_s^e \rangle \in \delta(q, (\pi, g_s^A, g_e^A))$ and $\pi' = update'((\pi^{**}, r_s, a_s^e))$.

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

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 $v_A \in B^{RA}$ be a valuation of the automaton
registers, and πs a partition of Rs. A partition π ∈ Π is consistent with νA and πs if there are register valuations ν0 ∈ DRs and νc ∈ DRc such that πs = f(νc) and π = f(νA ∪ νc). Thus, automata registers are related according to ν0 1, system registers are related according to πs, and environment registers are not related in any special way.

Theorem 10. Consider a reg-UPW A with Σ = Σf × ΣO, set of registers Ra, and an initial valuation ν0 1. Then, A is realizable by a kσ-register Σf/ΣO-transducer with a set of registers Rs with respect to environments that are kσ-register ΣO/Σf-transducers iff there is a partition πs of Rs and a (Σf × ΣO × Rs × RA)-transducer T′ such that for every partition π0 of R that is consistent with ν0 1 and πs, the transducer T′ realizes the (kσ, kσ, π0)-abstraction of A.

Proof. The theorem follows from the following claim. Fix a system kσ-register transducer Tsys with an initial valuation ν0 1 and fix an environment initial valuation ν0 c. Let A′ be the (ks, kσ, π0)-abstraction of A with π0 = f(ν0 1 ∪ ν0 c). Let Tsys be the register-less transducer corresponding to Tsys. Then, we have that Tsys |= A′ iff for every environment transducer Tenv with the initial valuation ν0 c, it holds that Tenv || Tsys |= A. The proof uses ideas similar to those in Theorem 7.

Assume first that there is Tenv initialized with ν0 c such that Tenv || Tsys ̸|= A. We then show that Tsys ̸|= A′. Let Tenv = (Sc, s′ 0 c, Re, νc) and Tsys = (Sf, s′ 0 f, Rs, ν0 c, τs). Let the computation Tenv||Tsys = ⟨⟨i0, o0⟩, i0, o0⟩⟨⟨i1, o1⟩, i1, o1⟩... induce the rejecting run ρA = ⟨q0, v0 1, q1, v1 1⟩... in A. Let ρc = ⟨s′ 0 c, ν0 c, s′ 1 c, ν1 c⟩... and ρs = ⟨s′ 0 f, ν0 1, s′ 1 f, ν1 f⟩... be the runs of interacting Tenv and Tsys, respectively. Let w′ sys be ⟨i0, ν0 c ∼ i0⟩⟨i1, ν1 c ∼ i1⟩... be an input word to Tsys. The input word w′ sys uniquely induces the computation wsys and the run ρsys = s0s1... We now define the rejecting run ρA′ on wsys. It starts in ⟨q0, π0, τ0⟩. Suppose that at moment j ≥ 0, the run ρA reached the configuration (q, νA), the run ρsys reached (s, νc), the run ρenv reached (s′, νc), and the run ρA′ reached (q, π, r). Assume that f(νc ∪ νA) = π. The assumption holds for j = 0. Later we will use the pair ⟨ρ′ c, o′⟩ ∈ ΣO × DR that is defined as follows. For j > 0, let ⟨ρ′ c, o′⟩ be the last output of the system transducer and then we assume that o′ = νc(r). For j = 0, let ⟨ρ′ c, o′⟩ be ⟨ς, νc(r)⟩.

Now, let Tenv in ρenv at the moment j transit from ⟨s, νc⟩ to ⟨s′, νc⟩, while reading ⟨ρ′ c, o′⟩ and outputting ⟨i, i⟩, whereas Tsys transits from ⟨s, νc⟩ to ⟨s′, νc⟩, while reading ⟨i, i⟩ and outputting ⟨o, o⟩. We need the following notations: g′ f = (νc ∼ o′), g′ f = (νc ∼ i), (a′ f, c′ e) = τs(s, o′, g′ f)Rνc × Rs, and (a′ f, c′ e) = τs(s, i, g′ f)Rνc × Rs. Note that i = νc(r) and o = ν′ c(r). Let ⟨q′ f, π′ e⟩ be a ⟨ς, νc⟩-successor of (q, νc) in ρA. We argue that ⟨ς, νc⟩ the pair ⟨q′ f, π′ e⟩ is a ⟨ς, νc⟩-successor of (q, π, r) in A′, where π′ = f(ν′ c ∪ ν′ c). Therefore, we apply this claim to get the successor ⟨q1, π1, r1⟩ of ⟨q0, π0, τ0⟩ in ρA, then apply it again to get ⟨q2, r2, r2⟩, and so on. In this way we construct the sought rejecting run ρA′ induced by the computation wsys. It is left to prove the claim †. By the definition of the A transitions, there exist g′ f, g′ f, aA such that the guard g′ f = (νA ∼ i), the guard g′ f = (νA ∼ o), and ⟨q′ f, aA⟩ ∈ δ(q, ⟨ς, νc⟩, g′ f, g′ f). Let π′ = update(π, r, a′ f) and π′ = update(π, r, a′ f). By Lemma 9, we have that π′ = f(ν′ c ∪ ν′ c ∪ ν′ c) and π′ = f(ν′ c ∪ ν′ c ∪ ν′ c). Now we need to show the following.

The guard g′ f is A-chosen by ⟨π′′, r⟩. This follows from Lemma 4(c).

The guard g′ f is A-chosen by ⟨π′′, r⟩. This follows from Lemma 4(c).

The satisfaction of δ is immediate because of the A transition.

π′ = f(ν′ c ∪ ν′ c ∪ ν′ c). By definition of δ′, the partition π′ = update(π′′, r, a′ f). By Lemma 9, update(π′′, r, a′ f) = f(ν′ c ∪ ν′ c ∪ ν′ c). The statement then follows.

This completes the proof of the first direction.
21:16  Register-Bounded Synthesis

Assume now that $T'_{sys} \not\models A'$. We show that there is $T_{env}$ initialized with $\nu_0$ such that $T_{env}||T_{sys} \not\models A$. Since $T'_{sys} \not\models A'$, there is an input word $w_{sys}'$ to $T'_{sys}$ that induces the run $\rho_{sys}' = s_0' \pi' \ldots$ and the computation $w_{sys}'$ which generates a rejecting run $\rho_A' = \langle q_0, \pi_0, r_0 \rangle (q_1, \pi_1, r_1) \ldots$ in $A'$. Let $\rho_{sys} = \langle s_0, (q_0, \pi_0, r_0) \rangle (s_1, (q_1, \pi_1, r_1)) \ldots$ be the corresponding run in the product of $T'_{sys}$ and $A'$. Wlog., we assume that $\rho_A$ is regular and hence can be expressed as a finite-length lasso. Thus, $\rho_A = x_0 \ldots x_{m-1}(x_m ... x_{m+n})^\omega$, where $x_0 ... x_{m-1}$ is the stem and $x_m ... x_{m+n}$ is the loop, where $m \geq n \geq 0$. Note that all elements $x_i$ are different. Given $w_{sys}'$, $\rho_{sys}'$, $\rho_A'$, and $\rho_A$, we construct $T_{env}$ initialized with $\nu_0$ such that $T_{env}||T_{sys}$ induces a rejecting run $\rho_A$ in $A$. We also construct $\rho_{sys}$, $\rho_{env}$, and $\rho_A$. The state space of $T_{env}$ is $S_e \subseteq S_s \times Q_{A'}$ and consists of the states that appear in $\rho_A$. The initial state $s_0 = \langle s_0, (q_0, \pi_0, r_0) \rangle$. Thus, we can write $\rho_{sys} = s_0' \ldots s_{m-1}' (s_{m} \ldots s_{m+n})^\omega$. The run $\rho_{sys}$ starts in $\langle s_0', \nu_0 \rangle$, the run $\rho_{env}$ starts in $\langle s_0', \nu_0 \rangle$, and the run $\rho_A$ starts in $\langle q, \nu_0 \rangle$. Suppose that at some moment $j \geq 0$, the run $\rho_{sys}$ reached state $s_s$, the run $\rho_{A'}$ reached state $\langle q, \pi, r \rangle$, the run $\rho_{sys}$ reached configuration $\langle s_s, \nu_s \rangle$, the run $\rho_{env}$ reached configuration $\langle s_e, \nu_e \rangle$, and the run $\rho_A$ reached configuration $\langle q, \nu_A \rangle$. Assume that $\pi = f(\nu_s \cup \nu_e \cup \nu_A)$ and $\sigma = (s_s, (q, \pi, r'))$. This holds for $j = 0$. We will use the pair $\langle o^*, \sigma^* \rangle \in Q_{\Sigma} \times D$ that is defined as follows. If $j > 0$, then $\langle o^*, \sigma^* \rangle$ is the output of $T_{sys}$ when it moved into $(s_s, \nu_s)$. If $j = 0$, let $\langle o^*, \sigma^* \rangle = (\sigma, (q_0, \nu_0(\nu_1)))$. Assume that for $j > 0$ the value $o^* = \nu_s(r)$. Now, let $T'_{sys}$ transit from $s$ into $s'$ while reading $\langle i, o_t \rangle$ and outputting $\langle o, \nu_t \rangle$. Let $A'$ transit into $\langle q', \pi', r_s \rangle$. Then, as we show later, the following holds $(\dagger \dagger)$: the environment transducer $T_{env}$ transits into $\langle s'_e, \nu'_e \rangle$ while reading $\langle o^*, \sigma^* \rangle$ and outputting $\langle i, o_t \rangle$, where $s'_e = \langle s'_e, (q', \pi', r_s) \rangle$, the system transducer $T_{sys}$ transits into $\langle s'_s, \nu'_s \rangle$ while reading $\langle i, o_t \rangle$ and outputting $\langle o, \nu_t \rangle$, where $o = \nu'_s(r_s)$, the automaton $A$ transits into $\langle q', \nu'_e \rangle$ on reading $\langle (i, o_t) \rangle$, and it holds that $\pi^* = f(\nu'_e \cup \nu'_o \cup \nu'_A)$. Using this claim, we iteratively construct the sought rejecting run $\rho_A$ of $A$ on $T_{env}||T_{sys}$, which completes the direction. Thus, we are left to prove the claim. The interesting part is the definition of the transition function $\tau_e$ of $T_{env}$, while the rest follows from Definitions and Lemmas 9 and 4(c). We now define $\tau_e$. Since $\langle q', \pi', r_s \rangle$ is a $\langle (i, o), g_A' \rangle$-successor of $\langle q, \pi, r \rangle$ in $A'$, there exist $a'_s, r_s$, and $a'_A$ such that $\pi^* = update(\pi, r, a'_s)$, $\pi^* = update(\pi, r, a'_s), \pi' = update(\pi^*, r_s, a'_A)$, $g_A'$ is $A$-chosen by $\langle \pi^*, r_s \rangle$, $g_A'$ is $A$-chosen by $\langle \pi^*, r_s \rangle$, and the guard $g_A'$ respects $r_s$ and $r_A$. First, if we visit $s_e$ for the first time (and thus $\tau_e(s_e, \sigma_e)$ was not defined yet), then we set $\tau_e(s_e, \sigma_e) = (s'_e, a'_e, i, r_s)$ for every $\sigma_e \in Q_{\Sigma} \times \mathbb{R}^R_s$. Second, suppose that we visited the state $s_e$ before. Because the run $\rho_A$ is a lasso, every state $s_e$ in $\rho_A$ has the unique successor state. Furthermore, the state $s_e$ uniquely identifies the letter that $A'$ is reading. Hence, the current letter $\langle (i, o), g'_e, r_s, a'_e \rangle$ and successor $\langle q', \pi', r_s \rangle$ of $A'$ are exactly as they were when we visited $s_e = \langle (s_e, (q, \pi, r)) \rangle$ for the first time. Hence, the previously set value of $\tau_e(s_e, \ldots)$, where _ is arbitrary, ensures that $T_{env}$ transits into a configuration with the state $s'_e$. Thus, the transition function $\tau_e$ moves $T_{env}$ as specified in claim $(\dagger \dagger)$.

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 $k_e$ 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 $O(nk^3)$ states and index $c$, where $k = k_s + k_e + k_A$. 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 $(k_s, k_e, \pi_0)$-abstractions of $A$, for every $\pi_0$ consistent with $\nu_0$ and $\pi_s$, are realized by a single transducer. There can be no more than $k_s^{k_e}$ 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 $\nu_0$. 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 $(k_s + k_c + k_A)^k_e$. 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 $(Nmc)^O(Nc) = (cnn(k^{k_e}))^{O(cn k^e)}$, 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 $k_{s}^{k_s}$, 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.

\[\text{Theorem 11.}\] 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 $(cnn(k_s + k_c + k_A))^{O(cn(k_s + k_c + k_A)(d_k + k_s + 1)))}$. Thus, it is polynomial in $m$, exponential in $c$, $n$, and $k_s$, and doubly-exponential in $k_A$ and $k_c$.

## Determinacy

For an $\omega$-regular specification $\psi$ over $(\Sigma \times D \times D)^{\omega}$, let $\neg \psi$ denote its complement, thus $L(\neg \psi) = (\Sigma \times D \times D)^{\omega} \setminus L(\psi)$. The specification $\psi$ may be given by a reg-UPW, yet the results in this section refer to general $\omega$-regular specifications over infinite alphabets. In particular, reg-UPWs are not closed under complementation. Traditional synthesis is determined, in the sense that for every $\omega$-regular specification $\psi$ over finite alphabets, either there is a system transducer that realizes $\psi$, or there is an environment transducer that realizes $\neg \psi$. Note that not having a system transducer that realizes $\psi$ only means that for every system $T_{sys}$, there is an environment transducer $T_{env}$ such that the computation $T_{env}||T_{sys}$ satisfies $\neg \psi$. This by itself does not imply that there is an environment $T_{env}$ such that for all systems $T_{sys}$, the computation $T_{env}||T_{sys}$ satisfies $\neg \psi$. However, by determinacy of Borel games [28], we know that the lack of $T_{sys}$ that realizes $\psi$ does imply the existence of $T_{env}$ that realizes $\neg \psi$. In [26], the authors show that determinacy no longer holds if we bound the number of states in systems or environments. In this section, we examine determinacy in a setting with a bounded number of registers.

For $k_s \in \mathbb{N}$ and $k_c \in \mathbb{N} \cup \{\infty\}$, we say that the register-bounded synthesis problem is $(k_s, k_c)$-determined if for every specification $\psi$, either

- $\psi$ is $I/O (k_s, k_c)$-realizable: there is a system transducer $T_{sys}$ with $k_s$ registers such that for all environment transducers $T_{env}$ with $k_c$ registers, the computation $T_{env}||T_{sys}$ satisfies $\psi$, or
- $\neg \psi$ is $O/I (k_c, k_s)$-realizable: there is an environment transducer $T_{env}$ with $k_c$ registers such that for all system transducers $T_{sys}$ with $k_s$ registers, the computation $T_{env}||T_{sys}$ satisfies $\neg \psi$.

In particular, the case $k_c = \infty$ corresponds to the setting where the number of registers in only one of the transducers is bounded. Note that $T_{sys}$ and $T_{env}$ are not completely dual: in both cases above we consider the computation $T_{env}||T_{sys}$, thus with $T_{env}$ moving first.

\[\text{Theorem 12.}\] For every $k_s \in \mathbb{N}$ and $k_c \in \mathbb{N} \cup \{\infty\}$, the register-bounded synthesis problem is not $(k_s, k_c)$-determined.
For every $k_s \geq 1$, we describe a specification $\psi_{k_s}$ such that for all $k_e \in \mathbb{N} \cup \{\infty\}$ neither $\psi_{k_e}$ is $I/O (k_s, k_e)$-realizable nor $\neg \psi_{k_e}$ is $O/I (k_e, k_s)$-realizable.

Let $\mathcal{D} = \mathbb{N}$. We describe $\psi_{k_s}$ by an LTL-like formalism, with the temporal operator $X$ (next) and Boolean propositions of the form $a = d$ and $i = d$, for $d \in \mathbb{N}$. The letter $i$ denotes the data-input to the system (that is, the data output of an environment), and $o$ denotes the data output of the system (that is, the data-input of the environment). For example, for $m \geq 2$, the formula $\varphi_m = (o = 3) \land X((o = 4) \land X((o = 5) \land \cdots \land X(o = m) \cdots))$ requires the system to output the sequence $3, 4, \ldots, m$ (when $m = 2$, the conjunction is empty, thus $\varphi_2 = \text{true}$).

Now, for $m \geq 1$, we define $\psi_m = (i = 1 \Rightarrow o = 2) \land (i = 2 \Rightarrow o = 1) \land X \varphi_{m+1}$. Thus, in order to satisfy $\psi_m$, a system has to output $x, 3, 4, \ldots, m + 1$, where $x$ depends on the input $i$ received from the environment in the first cycle: if $i = 1$, then $x = 2$, and if $i = 2$, then $x = 1$. Note that $\psi_m$ can be specified by a reg-UPW with $m + 1$ registers initialized to $1, \ldots, m + 1$.

Consider a bound $k_s \geq 1$. First, observe that for environment transducers $T_{env}$, there is a system transducer $T_{sys}$ with at most $k_s$ registers, such that the computation $T_{env} \parallel T_{sys}$ satisfies $\psi_{k_s}$. Indeed, $T_{sys}$ maintains the values $x, 3, 4, \ldots, k_s + 1$ in its registers and outputs them as required.

In addition, we argue that for all system transducers $T_{sys}$ with at most $k_s$ registers, there is an environment transducer $T_{env}$ with one register, such that the computation $T_{env} \parallel T_{sys}$ does not satisfy $\psi_{k_s}$. Essentially, this follows from the fact that the only data outputs that $T_{sys}$ can generate are stored in its registers, either in its initial register evaluation, or during the interaction, thus after being output by $T_{env}$. Thus, in order to output $x, 3, 4, \ldots, k_s + 1$, the transducer $T_{sys}$ has to store these values in its initial assignment. Indeed, it cannot count on $T_{env}$ to output them. Since $T_{sys}$ has only $k_s$ registers, and $k_s - 1$ of them are used for storing all values from $3, 4, \ldots, k_s + 1$, only one register can be used for storing the $x$. The value $x$, however, is not known in advance: for each value (in particular, 1 or 2) that $T_{sys}$ may store as the anticipated $x$, there is an environment transducer $T_{env}$ that provides as input the same value. Then, $\psi_{k_s}$ forces $T_{sys}$ to output the “dual” value, which is not stored in its registers.

\section{On Fixed, Finite, and Infinite Number of Registers}

For finite-state systems, finite-model properties assert that when a specification is given, there is often a cut-off point: an LTL formula $\psi$ is satisfiable iff there is a computation of length exponential in $|\psi|$ that satisfies it [15], and is realizable iff there is a transducer of size doubly-exponential in $|\psi|$ that realizes it [31]. In this section we study a similar question for register-bounded synthesis. We differentiate the strength of systems and environments with a fixed, finite, or infinite number of registers. In particular, we ask whether there is a bound on the number of system registers that may be required for the synthesis of a realizable specification, or a bound on the number of environment registers sufficient for model-checking systems against a given specification.

We first need some definitions and notations. Let $\mathcal{F}_k$, $\mathcal{F}_{FIN}$, and $\mathcal{F}_{INF}$ denote the classes of transducers with a fixed (to $k$), finite, and infinite number of registers, respectively. Similarly, let $\mathcal{A}_k$ and $\mathcal{A}_{FIN}$ denote the classes of reg-UPWs with with a fixed (to $k$) and finite number of registers. Note that $\mathcal{F}_{FIN} = \bigcup_k \mathcal{F}_k$ and $\mathcal{A}_{FIN} = \bigcup_k \mathcal{A}_k$.

For four parameters $k_s^1, k_s^2, k_e \in (\{\text{FIN}, \text{FIN}\} \cup \mathbb{N})$, and $k_s \in (\{\text{FIN}\} \cup \mathbb{N})$, we say that $k_s^2$-systems are at least as powerful as $k_s^1$-systems with respect to $k_e$-environments and $k_s$-specifications, denoted $k_s^2 \geq_{FIN}^s k_s^1$, if for every reg-UPW $A \in \mathcal{A}_{k_s^2}$, the existence of a system transducer in $\mathcal{T}_{k_s^2}$ that realizes $A$ against all environment transducers in $\mathcal{T}_{k_e}$ implies the
Theorem 13. The following relations among classes of systems and environments hold.

For all $k_e \in \{\{\text{INF}, \text{FIN}\} \cup \mathbb{N}\}$, $k_A \in \{\{\text{FIN}\} \cup \mathbb{N}\}$, and $k_s \in \mathbb{N}$, we have that

\[
\text{INF} >_{k_s,k_A} \text{FIN} >_{k_e,k_A} k_s + 1 >_{k_e,k_A} k_s.
\]

Let $k_s = \text{INF}$. Then, for all $k_A \in \{\{\text{FIN}\} \cup \mathbb{N}\}$ and $k_e \in \mathbb{N}$, we have that

\[
\text{INF} >_{k_s,k_A} \text{FIN} >_{k_e,k_A} k_e + 1 >_{k_e,k_A} k_e.
\]

Let $k_A = \text{FIN}$. Then, for all $k_s \in \{\{\text{FIN}\} \cup \mathbb{N}\}$ and $k_e \in \mathbb{N}$, we have that

\[
\text{INF} =_{k_s,k_A} \text{FIN} >_{k_e,k_A} k_e + 1 >_{k_e,k_A} k_e.
\]

For all $k_s \in \mathbb{N}$, $k_A \in \mathbb{N}$, and $0 < k_e < k_s + k_A$, we have that

\[
\text{INF} =_{k_s,k_A} \text{FIN} =_{k_e,k_A} k_s + k_A + 1 >_{k_e,k_A} k_e + 1 >_{k_e,k_A} k_e.
\]

Before proving the hierarchy theorem, we prove the following cut-off result about model checking of register transducers. A similar result is proven in [27, Thm.2], but we provide it here for completeness, using our model and notations.

Lemma 14. The following claims hold.

For every $T$ and $A$, if $T \not\models A$, then there exists an input word to $T$ with no more than $(k_s + k_A + 1)$ data values that induces a computation of $T$ rejected by $A$.

There exist $T$ and $A$ such that $T \not\models A$, but all input words with less than $(k_s + k_A + 1)$ data values generate computations of $T$ accepted by $A$.

Proof. We prove the first claim. Suppose that a $k_s$-register transducer $T = \langle \Sigma_I, \Sigma_O, S, s_0, R_A, v_0^T, \tau \rangle$ is rejected by a $k_A$-register UPW $A = \langle \Sigma, Q, q_0, R_A, v_0^A, \delta, \alpha \rangle$ where $\Sigma = \Sigma_I \times \Sigma_O$. Thus, there is an input word $w_I$ to $T$ that generates a computation $w = \langle (i_0, a_0), (i_0, a_0), (i_1, a_1), \ldots \rangle$ that induces a rejecting run $\rho_A = \langle q_0, v_0^A \rangle \langle q_1, v_1^A \rangle \ldots$ of $A$. Let $\rho_T = \langle (s_0, v_0^T), (s_1, v_1^T) \rangle \ldots$ be the run of $T$ on $w_I$. Using $w$, $\rho_T$, and $\rho_A$, we construct the data-domain $D'$ of size $k_s + k_A + 1$, and an input word $w'$ to $T$ with data values from $D'$, such that $w'$ generates a computation $w'$ in $T$ that induces a rejecting run $\rho'_A$ in $A$. The domain $D'$ is as follows. First, it contains all initial register values of $T$ and $A$. Second, we add new values to $D'$ until its size becomes $k_s + k_A + 1$.

We now construct $w'$, the run $\rho'_T$ of $T$ on $w'$, and $\rho'_A$. The run $\rho'_A$ starts in $\langle q_0, v_0^A \rangle$ and the run $\rho'_T$ starts in $\langle s_0, v_0^T \rangle$. As we show below, the following claim (†) holds. Suppose that...
in a step $j \geq 0$, the run $\rho_A$ reaches $\langle q, v \rangle$, the run $\rho'_A$ reaches $\langle q, w \rangle$, the run $\rho_T$ reaches $\langle s, v_o \rangle$, and the run $\rho'_T$ reaches $\langle s, w_o \rangle$. Assume that $f(v_A \cup v_o) = f(w_A \cup w_o)$. This holds initially, when $j = 0$. Assume that $T$ transits from $\langle s, v_o \rangle$ into $\langle s', v'_o \rangle$ on reading $(i, i)$ and outputs $(o, o)$. Assume that $A$ transits from $\langle q, v_A \rangle$ into $\langle q', v'_A \rangle$ on reading the letter $(i, o), i, o)$. Then, there exist $i^*, o^* \in D'$ such that $T$ transits from $\langle s, w_o \rangle$ to $\langle s', w'_o \rangle$ on reading $(i, i^*)$ and outputs $(o, o^*)$, the configuration $\langle q', w'_A \rangle$ is a $(\langle i, o \rangle, i^*, o^*)$-successor of $\langle q, w_A \rangle$, and $f(v'_A \cup v'_o) = f(v_A \cup w'_o)$. We can apply this claim in the initial step, when $j = 0$, and construct the first letter $(i, o, i^*, o^*)$ of the computation $w'$, the successor configuration $\langle q_1, w'_1 \rangle$ in $\rho'_A$, and the successor configuration $\langle s_1, w'_1 \rangle$ in $\rho'_T$. Note that $f(v'_1 \cup v'_o) = f(w'_1 \cup w'_o)$, and hence we can apply the claim again. By an iterative application, we construct the computation $w'$ and the rejecting run $\rho'_A$ of $A$ on $w'$. By projecting $w'$ into the inputs, we get the sought input word $w'_f$.

We now prove the claim (i). We first define $i^*$. Let $g_i = (v_i \sim i)$, $g_i^A = (v_A \sim i)$, and $g_i = g_i^T = (v_A \sim i)$. Recall that $g_i \subseteq R_A \cup R_A$. If $g_i \neq \mathcal{D}$, we set $i^* = (w_o \cup w_A)(r)$ for any $r \in g_i$, otherwise set $i^*$ to any value from $D'$ that is not used by $w_o \cup w_A$ (such a value exists since $|D'| > k_s + k_A$). Note that $(w_o \sim i^*) = (v_A \sim i^*) = g_i^A$ and $(w_A \sim i^*) = (v_A \sim i^*) = g_i$. We continue to define $o^*$. Let $s_i, r_s = s(i, i^*, o^*)$ be the storing mask used by $A$ during the transition in $\rho_T$ and in $\rho'_T$ (note that it is the same transition in $\rho_T$ and $\rho'_T$). Let $w'_A = \text{update}(w_o, i^*, o^*)$ be the updated transducer configuration in $\rho'_T$. Then, we set $o^* = w'_A(r_s)$. We note that $f(v'_A) = f(w'_A)$ by Lemma 5 and $(v_A \sim o^*) = (v_A \sim o)$ by Lemma 4(c). Let $g_i^A = (v_A \sim o)$. By now, we define $i^*, o^*$, the successor configuration $\langle s'_i, w'_o \rangle$ in $\rho'_T$, and now we proceed to define $w'_A$. Let $a_i^{A}$ be the storing mask used by $A$ when moving from $\langle q, v_o \rangle$ to $\langle q', v'_o \rangle$, hence $\langle q', a_i^{A} \rangle \in \delta(q, (i, o), g_i^T, g_i^A)$. Since $(w_A \sim o^*) = (v_A \sim o) = g_i^A$, we have that $\langle q, w_A \rangle$ is a $(\langle i, o \rangle, i^*, o^*)$-successor of $\langle q, w_A \rangle$, where $w'_A = \text{update}(w_A, i^*, o^*)$. Finally, to show that $f(w'_A \cup v'_o) = f(v'_1 \cup v'_o)$, we apply Lemma 5 twice. First, it gives that $f(w_A \cup w'_o) = f(w_A \cup w'_o)$; second, it gives the needed statement.

We now prove the second claim of the lemma. We construct the transducer $T$ with $k_s$ registers and the automaton $A$ with $k_A$ registers such that every input word to $T$ with less than $(k_s + k_A + 1)$ data values generates an accepted computation, and yet there is an input word with $(k_s + k_A + 1)$ data values that generates a rejected computation. Let $D = \mathbb{N}$, $\Sigma_D = \{a\}$ is the singleton, and $\Sigma_O = \{-b, b\}$. Consider the register transducer $T$ in the figure below. The registers $r^A_1, ..., r^A_k$ are initialized to the values $1, ..., k_s$, respectively. From the initial state, on reading the data-input that equals the value of register $r^A_1$, the transducer transits into the successor state, keeps the register values unchanged, outputs the data value of some non-important register $\_ \in R_A$ and the finite letter $-b$. On reading the data-input that is not equal to the value of $r^A_1$, the transducer transits into the sink state (now shown) where it stays forever and outputs the finite letter $-b$. The transitions from other states are similar. Thus, the transducer reaches the right-most state only if it reads the values $1, ..., k_s$. In the right-most state, the transducer loops forever regardless of what it reads, and outputs the finite letter $b$.

Now consider the reg-UCW below. The initial values of the automaton registers $r^A_1, ..., r^A_{k_s}$ are $k_s + 1, ..., k_s + k_A$, respectively. From the initial state, on reading the finite letter $-b$, the automaton loops, and does not change the register values. If the automaton reads the finite letter $b$ and the data-input $i$ equals the value of $r^A_1$, the automaton transits into the successor
state while keeping the register values intact. And so on. The transition into the right-most state \( \overline{b} \) requires the data-input \( b \) to differ from the values of all automaton registers. The state \( \overline{b} \) is a rejecting sink.

\[
\begin{align*}
&b \land i = r^A_1 \quad i = r^A_2 \quad \ldots \quad i = r^A_{k_A} \quad i \neq r^A_1 \land \ldots \land i \neq r^A_{k_A} \Rightarrow *
\end{align*}
\]

Now it is easy to see that in order to generate a rejecting computation of the transducer above, the input word has to begin with the data values \( 1, \ldots, k_s + k_A + 1 \), and every other data-word is accepted. This concludes the proof of the second claim, and of the lemma.

We now prove the theorem.

**Proof of Theorem 13.** We start with the relations between the system classes. Consider the specification “throughout the computation, all outputs have never appeared as inputs before” which can be expressed by a 1-register UPW. Consider an environment transducer that stores the current system output and sends it back to the system in the next round. Now, only system transducers with an infinite number of registers that initially store infinitely many values, can satisfy this specification when interacting with the above environment transducer.

Because the environment transducer and the UPW use only one register, we conclude that \( \text{INF} >^{\text{sys}}_{k_s,k_A} \text{FIN} \) for every \( k_s \in (\{\text{INF}, \text{FIN}\} \cup \mathbb{N}) \) and \( k_A \in (\{\text{FIN}\} \cup \mathbb{N}) \).

Consider a variation of the previous specification: “until (including) moment \( t \), all outputs have never appeared as inputs before”, where \( t > 0 \) is a parameter, which can also be specified by a 1-register UPW. Consider the same environment as before, namely the one that sends every system output back to the system. In order to realize the specification, a system transducer needs \( k_s \geq t \) registers. Hence, \( t + 1 >^{\text{sys}}_{k_s,k_A} t \) and \( \text{FIN} >^{\text{sys}}_{k_s,k_A} t \) for every \( k_s \in (\{\text{INF}, \text{FIN}\} \cup \mathbb{N}) \), \( k_A \in (\{\text{FIN}\} \cup \mathbb{N}) \), and \( t > 0 \).

We continue to the relations among the environment classes. Recall Lemma 14 that says that if a \( k_s \)-register system transducer \( T_{sys} \) has a computation rejected by a \( k_A \)-register UPW \( A \), then there is an input word to \( T_{sys} \) with no more than \((k_A + k_s + 1)\) data values that induces a computation of \( T_{sys} \), rejected by \( A \). This input word can be generated by an environment transducer with \((k_A + k_s + 1)\) registers. Then, it follows that \( \text{INF} =^{\text{env}}_{k_s,k_A} \text{FIN} \) for \( k_s, k_A \in (\{\text{FIN}\} \cup \mathbb{N}) \). Also, if \( k_s, k_A \in \mathbb{N} \), then \( \text{INF} =^{\text{env}}_{k_s,k_A} \text{FIN} =^{\text{env}}_{k_s,k_A} k_s + k_A + 1 \).

Because the bound \((k_A + k_s + 1)\) is tight (the second claim in Lemma 14), for some specifications \( A \) and system transducers \( T_{sys} \) that do not realize \( A \), we have that \( T_{sys} \) does realize \( A \) with respect to all environments with \( k_s + k_A \) registers. Hence, \( k_s + k_A + 1 >^{\text{env}}_{k_s,k_A} k_s + k_A + 1 \) for every \( k_s < k_s + k_A \). The same consideration also implies that \( \text{FIN} >^{\text{env}}_{k_s,k_A} k_s \) for every \( k_s \in (\{\text{INF}, \text{FIN}\} \cup \mathbb{N}) \) and \( k_A \in (\{\text{FIN}\} \cup \mathbb{N}) \).

Finally, consider a letter \( a \in \Sigma_O \) and consider the specification “eventually always \( a \)”. Consider a system that stores every data-input into its registers. Further, the system outputs \( a \) whenever it reads a data-input that has appeared before (and hence equals to one of its registers). Consider an environment, with an infinite number of registers, that always provides a new value to the system. When interacting with this environment, the system does not satisfy the specification. On the other hand, every environment transducer with a finite number of registers starts, at some point, to repeat its values. When interacting with such an environment transducer, the system satisfies the specification. Hence, \( \text{INF} >^{\text{env}}_{k_s,k_A} \text{FIN} \) for \( k_s = \text{INF} \) and every \( k_A \in (\{\text{FIN}\} \cup \mathbb{N}) \).

Theorem 13 implies, in particular, that once the numbers \( k_s \) and \( k_A \) of system and automaton registers are fixed, then for model checking it is sufficient to consider environments
with \((k_s + k_A + 1)\) registers only. Also, when \(k_s\) is not fixed, such a cut-off does not exist. A similar question is whether there exists a cut-off for synthesis, once the numbers \(k_s\) and \(k_A\) are fixed. Unfortunately, the answer is negative. Indeed, such a bound would lift the decidability of the environment-system-bounded synthesis problem (Theorem 10) to the synthesis problem of unbounded systems wrt. bounded environments, and the latter is undecidable (by the same argument as Theorem 3). Note, however, that Theorem 3 considers system transducers initialized with unboundedly many data values. This leaves a hope for a positive answer to the system cut-off question for the case when system transducers have a finite-but-unbounded number of registers initialized with a fixed number of data values, and environment transducers have a fixed number of registers. We leave this question for future work. Finally, we note that the variant of a synthesis problem, where system transducers have a finite-but-unbounded number of registers initialized to the same value, and environment transducers have a finite-but-unbounded number of registers with no restriction on initial values, is undecidable [16].

References


Register-Bounded Synthesis

