On Synthesis of Specifications with Arithmetic

Rachel Faran and Orna Kupferman

The Hebrew University of Jerusalem, Israel
{rachelmi, orna}@cs.huji.ac.il

Abstract. *Variable automata with arithmetic* enable the specification of reactive systems with variables over an infinite domain of numeric values and whose operation involves arithmetic manipulation of these values [10]. We study the *synthesis problem* for such specifications. While the problem is in general undecidable, we define a fragment, namely *semantically deterministic* variable automata with arithmetic, for which the problem is decidable. Essentially, an automaton is semantically deterministic if the restrictions on the possible assignments to the variables that are accumulated along its runs resolve its nondeterministic choices. We show that semantically deterministic automata can specify many interesting behaviors – many more than deterministic ones, and that the synthesis problem for them can be reduced to a solution of a two-player game. For automata with simple guards, the game has a finite state space, and the synthesis problem can be solved in time polynomial in the automaton and exponential in the number of its variables.

1 Introduction

Synthesis is the automated construction of systems from their specifications [6, 20]. The specification is typically given by a temporal-logic formula or an automaton, and it 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. Since its introduction, synthesis has been one of the most studied problems in formal methods, with extensive research on wider settings, heuristics, and applications [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 over finite alphabets. 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 [17], software with integer parameters [3], communication protocols with message parameters [7], datalog systems with infinite data domain [24], and many more [5, 4]. Lifting automata-based methods to the setting of such systems requires the introduction of automata with *infinite alphabets*. The latter include *registers* [23], *pebbles* [18], *variables* [11], and *data* [2] automata. These formalisms refer to the infinite values by comparing them to each other. Thus, the exact value is abstracted: one can specify, for example, that each value received as input is generated as an output at

least once during the next 10 transitions, but cannot specify, for example, that if a value $x \in \mathbb{Q}$ is received as input then the next 10 transitions include only outputs of values in $[x-5, x+5]$.

In [10], we introduced automata with arithmetic, which do support specifications as the latter. Here, we consider *nondeterministic looping word automata with arithmetic* (NLWAs, for short), which define languages over alphabets of the form $\Sigma \times \mathbb{Q}$, for some finite set Σ. Each NLWA has a finite set X of variables over $\mathbb{Q}$. The transitions of an NLWA are labeled by both letters from Σ and guards involving values in $\mathbb{Q}$, variables in X, and the symbol $\star$, which refers to the $\mathbb{Q}$-value of the letter read. A word w is accepted by an NLWA $\mathcal{A}$ if there is an assignment to the variables that appear in $\mathcal{A}$ such that there is an infinite run of $\mathcal{A}$ on w. In particular, all the guards along the run are satisfied. The *looping* acceptance condition is a special case of the Büchi condition. It captures *safety properties*, and we use it here in order to circumvent technical challenges that are irrelevant for the challenge of handling arithmetic. It is shown in [10] that many decision problems on NLWAs are decidable, essentially by replacing queries about reachability via runs by queries about the satisfaction of guards accumulated during runs.

Recent work on synthesis shows that while the synthesis problem for register automata is undecidable [8], the *register-bounded* synthesis problem is decidable [14, 13, 9]. There, the input to the problem includes bounds on the number of registers of the system and/or the environment. The bounds enable an abstraction of the exact values stored in the registers by their partition into equivalence classes. The abstraction, however, strongly depends on the fact that the only operation that register automata apply to the stored values is comparison. As discussed above, such comparisons cannot handle specifications that refer to the values, in particular ones with arithmetic.

We study the synthesis problem for NLWAs and point to a decidable fragment. In the setting of finite alphabets, the specifications are over an alphabet $2^{I \cup O}$, for finite sets I and O of input and output signals. The synthesis problem for specifications given by deterministic automata is solvable by a reduction to a two-player game [1]. The positions of the game are the states of the automaton. In each round of the game, one player (the environment) provides an input – a letter in 2^I, the second player (the system) responds with an output – a letter in 2^O, and the game transits to the corresponding successor state. The system wins the game if, no matter which inputs the environment provides, the interaction between the players generates a computation along with an accepting run of the automaton on it. When the automaton is nondeterministic, the system responds not only with an output, but also with a transition that should be taken. This is problematic, as this choice of a transition should accommodate all possible future choices of the environment. In particular, if different future choices of the environment induce computations that are all in the language of the automaton yet require different nondeterministic choices, the system cannot win.

The need to work with deterministic automata is a known barrier for synthesis in practice: algorithms involve complicated determinization constructions [21] or acrobatics for circumventing determinization [16]. In the case of automata with arithmetic, the challenge is bigger: First, even when the automata are deterministic, the strategies of the players depend on values in $\mathbb{Q}$, thus the game has infinitely many configurations. Second, determinism significantly reduces the expressive power of NLWAs. Indeed, it

requires that for every state q and letter $\sigma \in \Sigma$, at most one guard in all the σ-transitions from q is satisfiable. For example, we cannot have two σ-transitions from q, one guarded by $x > 8$ and the second by $x \leq 5$. We suggest to distinguish between three levels of determinism in an NLWA $\mathcal{A}$. In addition to the standard definition, by which $\mathcal{A}$ is deterministic (DLWA) if it has at most one run on every word, we say that $\mathcal{A}$ is *deterministic per assignment* (DPA-NLWA) if for every assignment to the variables in X, there is at most one run of $\mathcal{A}$ on every word. Then, $\mathcal{A}$ is *semantically deterministic* (SD-NLWA) if the restrictions on the possible assignments to the variables in X that are accumulated along the run resolve its nondeterministic choices. In the example above, if every run that leads to the state q is such that at most one of $\gamma \wedge (x > 8)$ or $\gamma \wedge (x \leq 5)$ is satisfiable, where γ is the the conjunction of guards accumulated during the run, then there is no real choice to make when σ is read in state q.

We show that while DPA-NLWAs are as expressive as NLWAs, they are not useful in synthesis. Indeed, the assignment to the variables in X in the different runs on $\mathcal{A}$ is not known in advance to the system and the environment, and may be different for the different computations they generate. On the other hand, SD-NLWAs can be soundly used in the synthesis game. Intuitively, as has been the case with *good for games* (GFG) automata [15, 12], SD-NLWAs can resolve their nondeterministic choices in a way that only depends on the past. While in GFG automata, resolving of nondeterminism concerns the limit behavior of the run, in SD-NLWAs it also concerns the restrictions on the possible assignment to the variables in X. Consequently, while GFG automata with an acceptance condition γ (e.g., Büchi) are as expressive as deterministic γ automata [15, 19], SD-NLWAs are strictly more expressive than DLWAs. Moreover, we argue that natural NLWAs are sematically deterministic. Indeed, in a typical specification, one first assigns values into the variables in the specification and then resolves guards that depend on the assignments. In the example above, it is typically the case that paths to the a state q with branches as above include guards that compare x with $\star$, namely with the $\mathbb{Q}$-element of one of the letters in the input, which restricts possible assignments to x in a way that makes only one of $(x > 8)$ or $(x \leq 5)$ satisfiable.

We solve the synthesis problem for SD-NLWAs by reducing it to a two-player game. While semantic determinism handles the nondeterminism, there is also the challenge of the infinite variable domain. In order to obtain a finite game, we show that when the guards of the SD-NLWA are simple, namely each term refers to at most one variable or to $\star$, then we can exploit the density of $\mathbb{Q}$ and abstract the infinitely many values in $\mathbb{Q}$ by finitely many partitions of $\mathbb{Q}$ and orders on X induced by the guards in the SD-NLWAs. The transducers induced by a winning strategy use the same set X of variables, and are also semantically deterministic, in the sense that guards over the input values are used to resolve nondeterminism. The game, and hence also the synthesis problem, can be solved in time polynomial in the SD-NLWA and exponential in X.

2 Preliminaries

For a finite set X of variables, the set of *terms over* X, denoted Θ_X, is defined inductively as follows.

- m, x, and $\star$, for $m \in \mathbb{Q}$, $x \in X$, and the symbol $\star$.

– $t_1 + t_2$ and $t_1 - t_2$, for $t_1, t_2 \in \Theta_X$.

A term is *simple* if it contains at most one element in $X \cup \{\star\}$. For a number $k \in \mathbb{Q}$ and an assignment $f : X \to \mathbb{Q}$, let $f^k : X \cup \{\star\} \to \mathbb{Q}$ be an extension of f where for all $x \in X$, we have $f^k(x) = f(x)$, and for the symbol $\star$ we have $f^k(\star) = k$. Given k and f, we can extend f^k to terms over X in the expected way; for example, $f^k : \Theta_X \to \mathbb{Q}$ is such that $f^k(t_1 + t_2) = f^k(t_1) + f^k(t_2)$.

The set of *guards over* X, denoted $\mathcal{G}_X$, is defined inductively as follows.

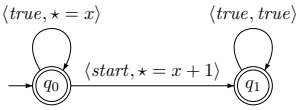
– $t_1 < t_2$ and $t_1 = t_2$, for $t_1, t_2 \in \Theta_X$,
– $\neg\gamma_1$ and $\gamma_1 \wedge \gamma_2$, for $\gamma_1, \gamma_2 \in \mathcal{G}_X$.

For $k \in \mathbb{Q}$, an assignment $f : X \to \mathbb{Q}$, and a guard $\gamma \in \mathcal{G}_X$, we define when k *satisfies* γ *under* f, denoted $k \models_f \gamma$, by induction on the structure of γ as follows.

– For two terms $t_1, t_2 \in \Theta_X$, we have that $k \models_f (t_1 < t_2)$ iff $f^k(t_1) < f^k(t_2)$, and $k \models_f (t_1 = t_2)$ iff $f^k(t_1) = f^k(t_2)$.
– For guards $\gamma_1, \gamma_2 \in \mathcal{G}_X$, we have that $k \models_f \neg\gamma$ if $k \not\models_f \gamma$, and $k \models_f \gamma_1 \wedge \gamma_2$ if $k \models_f \gamma_1$ and $k \models_f \gamma_2$.

Using Boolean operations, we can compare terms also by the $\leq$, $\geq$, and $>$ relations. We refer to guards of the form $t_1 \sim t_2$, for $\sim \in \{\leq, \geq, =, <, >\}$ as *atomic guards*. As it is useless to have $\star$ in both t_1 and t_2, we assume that when an atomic guard includes $\star$, then $\star$ appears only in t_1. A guard is *simple* if all its terms are simple. For example, $x_1 \leq x_2 + 5$ and $\star = x_1$ are simple, whereas $x_1 + x_2 \leq 5$ is not.

We are interested in languages of infinite words in $(\Sigma \times \mathbb{Q})^\omega$. A *nondeterministic looping word automaton with arithmetic* (NLWA, for short) is a tuple $\mathcal{A} = \langle \Sigma, X, Q, Q_0, \Delta \rangle$, where Σ is an alphabet, X is a set of variables, Q is a set of states, $Q_0 \subseteq Q$ is a set of initial states, and $\Delta \subseteq Q \times \Sigma \times \mathcal{G}_X \times Q$ is a transition relation. Thus, each transition is labeled by both a letter in Σ and a guard in $\mathcal{G}_X$. A *run* of $\mathcal{A}$ on an infinite word $\langle \sigma_0, k_0 \rangle, \langle \sigma_1, k_1 \rangle, \ldots$ over $\Sigma \times \mathbb{Q}$ is a sequence of states $q_0, q_1, \ldots$, where $q_0 \in Q_0$, and there is an assignment $f : X \to \mathbb{Q}$ such that for every position $i \geq 0$, there is a transition $\langle q_i, \sigma_i, \gamma, q_{i+1} \rangle \in \Delta$ such that $k_i \models_f \gamma$. Note that the assignment f is fixed throughout the run, and that all the runs are infinite. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of all words $w \in (\Sigma \times \mathbb{Q})^\omega$ such that there is a run of $\mathcal{A}$ on w. For example, the NLWA below accepts all the words in which all the letters agree on the $\mathbb{Q}$-component, possibly only until the Σ-component is *start* and the $\mathbb{Q}$-component is increased by 1.



We turn to define determinism for NLWAs. Consider an NLWA $\mathcal{A} = \langle \Sigma, X, Q, Q_0, \Delta \rangle$. We say that $\mathcal{A}$ is *deterministic per assignment* (DPA-NLWA, for short) if $|Q_0| = 1$, and for every function $f : X \to \mathbb{Q}$, state $q \in Q$, and letter $\langle \sigma, k \rangle \in \Sigma \times \mathbb{Q}$, there is at most one transition $\langle q, \sigma, \gamma, q' \rangle \in \Delta$ such that $k \models_f \gamma$. In other words, $\mathcal{A}$ is deterministic per assignment if for every assignment to X, there is at most one run of $\mathcal{A}$ on every word in $(\Sigma \times \mathbb{Q})^\omega$. We say that $\mathcal{A}$ is *deterministic* (DLWA, for short) if the choice of

the transition does not depend on the assignment to X. Formally, we have that $\mathcal{A}$ is deterministic if $|Q_0| = 1$, and for every state $q \in Q$ and letter $\langle \sigma, k \rangle \in \Sigma \times \mathbb{Q}$, there is at most one transition $\langle q, \sigma, \gamma, q' \rangle \in \Delta$ such that there exists $f : X \to \mathbb{Q}$ for which $k \models_f \gamma$.

Finally, an NLWA is *semantically deterministic* (SD-NLWA, for short) if for every state q and every run r that reaches q, the restrictions on the variable values accumulated throughout r resolve the nondeterminism in q. Formally, for a finite word $w = \langle \sigma_0, k_0 \rangle, \ldots \langle \sigma_t, k_t \rangle \in (\Sigma \times \mathbb{Q})^*$, let $r = q_0, q_1, q_2, \ldots, q_{t+1}$ be a run on w, and let γ_i be the guard that labels the σ_i-transition from q_i to q_{i+1}. We denote $\gamma^r = \bigwedge_{1 \leq i \leq t} \gamma_i$. We say that $\mathcal{A}$ is semantically deterministic if $|Q_0| = 1$, and for every state $q \in Q$, every run r from q_0 to q, and every letter $\langle \sigma, k \rangle \in \Sigma \times \mathbb{Q}$, there is at most one transition $\langle q, \sigma, \gamma, q' \rangle \in \Delta$ such that there exists $f : X \to \mathbb{Q}$ for which $k \models_f \gamma \wedge \gamma^r$.

In the context of open systems, we define automata over $\Sigma = 2^{I \cup O}$ for finite sets I and O of input and output signals, respectively. Also, rather than a single value in $\mathbb{Q}$, we let each position in the computation include two such values – input and output. Thus, a computation is $\pi = \langle \sigma_0, k_0^I, k_0^O \rangle, \langle \sigma_1, k_1^I, k_1^O \rangle, \cdots \in (2^{I \cup O} \times \mathbb{Q} \times \mathbb{Q})^\omega$. In order to indicate whether a guard refers to the input or output value, we use a variant of NLWAs, termed NLWAs^{IO}, in which the $\star$ in the guards is parameterized by the letters I and O. Thus, the atomic guards in an NLWA^{IO} with variables in X include $\star_I \sim t$ and $\star_O \sim t$, for a term $t \in \Theta_X$ and $\sim \in \{\leq, \geq, =, <, >\}$. Then, for a pair of numbers $\langle k^I, k^O \rangle \in \mathbb{Q} \times \mathbb{Q}$, we have that $\langle k^I, k^O \rangle$ satisfies $\star_I \sim t$ iff $k^I \sim t$. Likewise, $\langle k^I, k^O \rangle$ satisfies $\star_O \sim t$ iff $k^O \sim t$. The semantics of NLWAs^{IO} is similar to that of NLWAs, except that for $\gamma \in \mathcal{G}_X$, an assignment $f : X \to \mathbb{Q}$ and a position $j \geq 0$, we have that $(\pi, j) \models_f \gamma$ iff $\langle k_j^I, k_j^O \rangle \models_f \gamma$. Note that the notions of determinism, determinism per assignment and semantic determinism can be easily extended to NLWAs^{IO}.

For finite sets I and O of input and output signals, respectively, a finite-state I/O *transducer over* $\mathbb{Q}$ is $\mathcal{T} = \langle I, O, S, s_0, \mathcal{G}^I, \mathcal{G}^O, \rho, \tau \rangle$, where S is a finite set of states, $s_0 \in S$ is an initial state, $\mathcal{G}^I, \mathcal{G}^O \subset \mathcal{G}_X$ are finite sets of atomic guards that may include $\star_I$ and $\star_O$, respectively, $\rho : S \times 2^I \times \mathcal{G}^I \to S$ is a transition function, and $\tau : S \to 2^O \times \mathcal{G}^O$ is a labelling function on the states. Note that $\mathcal{T}$ abstracts the concrete input and output values, and partitions the infinitely many values according to satisfaction of guards in $\mathcal{G}^I$ and $\mathcal{G}^O$.

Intuitively, $\mathcal{T}$ models the interaction of an environment that generates at each moment in time a letter in 2^I and a value in $\mathbb{Q}$ with a system that responds with a letters in 2^O and a value in $\mathbb{Q}$. Let $\mathcal{F} = \{F : F \subseteq \mathbb{Q}^X\}$, and consider an input word $w = \langle i_0, k_0^I \rangle \cdot \langle i_1, k_1^I \rangle \cdots \in (2^I \times \mathbb{Q})^\omega$. A *run* of $\mathcal{T}$ on w is a sequence $\langle s_0, o_0, k_0^O, F_0 \rangle$, $\langle s_1, o_1, k_1^O, F_1 \rangle, \langle s_2, o_2, k_2^O, F_2 \rangle \ldots \in (S \times 2^O \times \mathbb{Q} \times \mathcal{F})^\omega$, where the S-components describe the states visited along the run, the 2^O- and $\mathbb{Q}$-components describe the outputs, and the $\mathcal{F}$-components describe the set of possible assignments to the variables in X that are consistent with the restrictions accumulating during the run – restrictions imposed by both the guards along the transitions and their combination with the value of the input variables and the assignments guards in the states and their combination with the value of the output variables. Accordingly, $F_0 = \mathbb{Q}^X$, and for all $j \geq 0$, we have that $s_{j+1} = \rho(s_j, i_j, \gamma_j^I)$ for γ_j^I such that $k_j^I \models_f \gamma_j^I$ for some $f \in F_j$, $\tau(s_j) = \langle o_j, \gamma_j^O \rangle$ for γ_j^O such that $k_j^I \models_f \gamma_j^I$ for some $f \in F_j \cap \{f \in \mathbb{Q}^X : k_j^I \models_f \gamma_j^I\}$, and $F_{j+1} =$

$F_j \cap \{f \in \mathbb{Q}^X : k_j^I \models_f \gamma_j^I\} \cap \{f \in \mathbb{Q}^X : k_j^O \models_f \gamma_j^O\}$. We require ρ to be *receptive* and *deterministic*, in the sense that for every input word $w = \langle i_0, k_0^I\rangle \cdot \langle i_1, k_1^I\rangle \cdots \in (2^I \times \mathbb{Q})^\omega$ and $j \geq 0$, there is exactly one state $s_{j+1} = \rho(s_j, i_j, \gamma_j^I)$ such that $k_j^I \models_f \gamma_j^I$ for some $f \in F_j$. An *output* of $\mathcal{T}$ on w is $\langle o_1, k_1^O\rangle \cdot \langle o_2, k_2^O\rangle \cdots \in (2^O \times \mathbb{Q})^\omega$ such that there is a run $\langle s_0, o_0, k_0^O, F_0\rangle, \langle s_1, o_1, k_1^O, F_1\rangle, \langle s_2, o_2, k_2^O, F_2\rangle \ldots$ of $\mathcal{T}$ on w. Note that the first output assignment is that of s_1, thus $\tau(s_0)$ is ignored. This reflects the fact that the environment initiates the interaction. A *computation of $\mathcal{T}$ on w* is then $\mathcal{T}(w) = \langle i_0 \cup o_1, k_0^I, k_1^O\rangle, \langle i_1 \cup o_2, k_1^I, k_2^O\rangle, \ldots \in (2^{I\cup O} \times \mathbb{Q} \times \mathbb{Q})^\omega$.

For an NLWA^{IO} $\mathcal{A}$, we say that $\mathcal{T}$ *realizes* $L(\mathcal{A})$ if for every input word $w \in (2^I \times \mathbb{Q})^\omega$, all the computations of $\mathcal{T}$ on w are in $L(\mathcal{A})$. The *synthesis* problem for NLWA^{IO} is then to decide, given an NLWA^{IO} $\mathcal{A}$, whether $L(\mathcal{A})$ is realizable, and if so, to return an I/O-transducer that realizes it.

3 Different Levels of Nondeterminism in NLWAs

In this section we study the different levels of nondeterminism in NLWAs. We start with the expressive power of DPA-NLWAs and SD-NLWAs, with respect to NLWAs, DLWAs, and each other, and continue to the problem of deciding the nondeterminism level of a given automaton.

We first prove that determinism per assignment does not restrict the expressive power of NLWAs. Thus, DPA-NLWAs are as expressive as NLWA. The proof is constructive and the idea is based on an elaboration of the subset construction. There, given a nondeterministic automaton $\mathcal{A}$ with state space Q, a deterministic equivalent automaton $\mathcal{A}'$ has state space 2^Q, and the transitions are defined so that $\mathcal{A}'$ visits a state $S \in 2^Q$ after reading a prefix w if S is the set of states that $\mathcal{A}$ may reach in at least one run after reading w. Since the path traversed by $\mathcal{A}$ when it reads w is not important (recall we consider looping automata), the subset construction maintains all the information needed. In the case of NLWAs, the paths traversed are important – they induce restrictions on possible assignments to X. Accordingly, an adoption of the subset construction to DPA-NLWAs involves a duplication of the set of variables – one copy for each transition. Then, the state space of the equivalent DPA-NLWA consists of subsets of Q along with an indication, for each $x \in X$, which copies of x should be assigned the same value. The detailed construction appears in Appendix A.1.

Theorem 1. *DPA-NLWAs are as expressive as NLWAs.*

Theorem 1 is quite surprising, but is of no real help in the context of synthesis. Indeed (see formal proof in Lemma 1), determinization per assignment is not useful when we run the automaton simultaneously on all the computations of a transducer, as different computations may be accepted with different assignments. Accordingly, we turn to focus on semantically-deterministic NLWAs. As we show in Lemma 2, this model of determinism is useful for solving the synthesis problem.

Theorem 2. *SD-NLWAs are strictly less expressive than NLWAs.*

Proof. The NLWA $\mathcal{A}$ in the left of Figure 1 accepts all the words in which the projection on the Σ-component is in $a^\omega + a^* \cdot b^\omega$, where in the second case the first b comes

after two letters that agree on their $\mathbb{Q}$-component. In Appendix A.2, we show that $\mathcal{A}$ does not have an equivalent SD-NLWA. Intuitively, it follows from the fact that the nondeterministic choice between a^ω and $a^* \cdot b^\omega$ should be taken before x is assigned a value. □

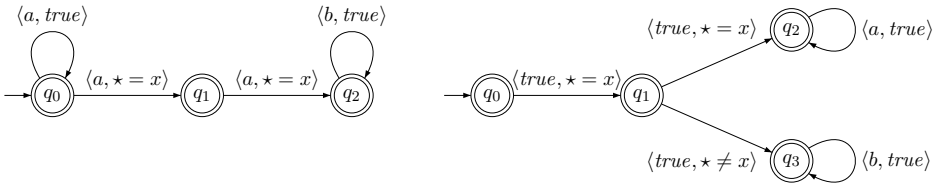


Fig. 1. An NLWA with no equivalent SD-NLWA, and an SD-NLWA with no equivalent DLWA.

Theorem 3. *SD-NLWAs are strictly more expressive than DLWAs.*

Proof. Consider the NLWA $\mathcal{A}$ in the right of Figure 1. It is easy to see that $\mathcal{A}$ is an SD-NLWA. Indeed, when a run reaches q_1, the variable x is already assigned a value, thus the nondeterminism in q_1 can be resolved, and all the other states have exactly one outgoing transition. Assume by way of contradiction that there is a DLWA $\mathcal{A}'$ that recognizes $L(\mathcal{A})$. By the pigeonhole principle, there are two numbers $k_1 \neq k_2$ such that $\mathcal{A}'$ reaches the same state after reading either $\langle a, k_1\rangle$ or $\langle a, k_2\rangle$. Since $L(\mathcal{A}) = L(\mathcal{A}')$, we have that the run of $\mathcal{A}'$ on $\langle a, k_1\rangle^\omega$ is accepting. Since $\mathcal{A}'$ is deterministic, we have that this run is also the run of $\mathcal{A}'$ on $\langle a, k_2\rangle \cdot \langle a, k_1\rangle^\omega$, which is not in $L(\mathcal{A})$. □

We turn to discuss the problem of deciding the type of a given automaton. Note that we consider the syntactic questions, namely whether the given automaton is deterministic per assignment or semantically deterministic, and not the semantic one, namely whether it has an equivalent DPA-NLWA or SD-NLWA.

Theorem 4. *The problems of deciding whether a given NLWA is a DPA-NLWA or is an SD-NLWA are co-NP-complete.*

Proof. Given an NLWA $\mathcal{A} = \langle \Sigma, X, Q, Q_0, \Delta\rangle$, a nondeterministic Turing machine can decide in polynomial time that $\mathcal{A}$ is not a DPA-NLWA by guessing a reachable state $q \in Q$, a letter $\langle \sigma, k\rangle \in \Sigma \times \mathbb{Q}$, and two transitions $\langle q, \sigma, \gamma_1, q'\rangle, \langle q, \sigma, \gamma_2, q''\rangle$ such that $k \models_f \gamma_1 \wedge \gamma_2$ for some $f : X \to \mathbb{Q}$. Deciding that $\mathcal{A}$ is not an SD-NLWA can be done by guessing, in addition, two assignments $f_1, f_2 : X \to \mathbb{Q}$, a finite word $w \in (\Sigma \times \mathbb{Q})^*$, and a run r on it from the initial state q_0 to q, where all the guards throughout r are satisfiable by the corresponding letters in w under both f_1 and f_2. Then, the letter $\langle \sigma, k\rangle$ and the transitions $\langle q, \sigma, \gamma_1, q'\rangle, \langle q, \sigma, \gamma_2, q''\rangle$ should be such that $k \models_{f_1} \gamma_1$ and $k \models_{f_2} \gamma_2$.

By [22], a solution to a linear inequalities system is of size polynomial in the size of the system. In addition, the problem of checking satisfiability of a Boolean formula over

a set of linear inequalities is P-reducible to the problem of solving a linear inequalities system. Note that by conjuncting the guards throughout a finite path in $\mathcal{A}$, we get a Boolean formula over the guards in $\mathcal{A}$ that is satisfiable iff the path is a run on some word. Therefore, both f_1 and f_2 are of polynomial size, and the upper bounds follow.

For the lower bounds, consider a Boolean formula φ over a set $X = \{x_1, \ldots, x_n\}$ of variables and the NLWA $\mathcal{A}$ described in Figure 2. Note that the only state in $\mathcal{A}$ in which nondeterminism may appear is q_n, and that φ is satisfiable iff both of the edges that leave q_n are satisfiable. Thus, $\mathcal{A}$ is a DPA-NLWA iff φ is not satisfiable. Since every path from q_0 to q_n induces an assignment $f : X \to \{0, 1\}$, we also have that $\mathcal{A}$ is an SD-NLWA iff φ is not satisfiable, and the lower bounds follow. □

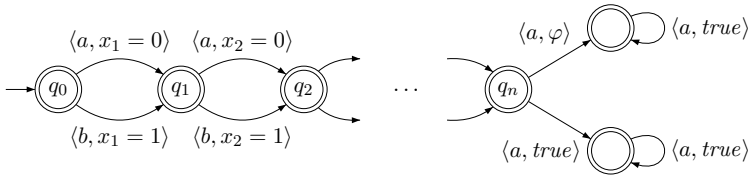


Fig. 2. The NLWA $\mathcal{A}$ is an SD-NLWA and a DPA-NLWA iff φ is not satisfiable.

4 Synthesis

Recall that in synthesis, the goal is to decide, given an automaton $\mathcal{A}$ over $2^{I \cup O}$, whether $L(\mathcal{A})$ is realizable, and if so, to return an I/O-transducer that realizes it. As described in Section 1, the synthesis problem is reduced to deciding the winner in a two-player game that is played over $\mathcal{A}$. For finite-state deterministic automata, the game is finite and can be decided in polynomial time. In this section we study the synthesis problem for NLWAs^{IO}. We first define the synthesis game for them, then show that it is not helpful for synthesis of general NLWAs^{IO} and even DPA-NLWAs^{IO}, but is helpful for SD-NLWAs^{IO}. We then describe an algorithm for solving the synthesis game induced by SD-NLWAs^{IO} all whose guards are simple. The complexity of the algorithm is polynomial in the state space of the automaton and exponential in the number of variables. We conclude that the synthesis problem for SD-NLWAs^{IO} with simple guards is decidable with the above complexity.

4.1 The synthesis game

Consider a NLWA^{IO} $\mathcal{A} = \langle 2^{I \cup O}, X, Q, q_0, \Delta \rangle$. The players in the *synthesis game* $G^{\mathcal{A}}$ are OR and AND (the system and the environment players, respectively), its possible locations are $Q \cup \{\bot\}$, and its initial location is q_0. Let q_j be the location of the game at the start of the j-th round. The j-th round of a play consists of two parts: first, AND chooses a letter $\langle i_j, k^I_j \rangle \in 2^I \times \mathbb{Q}$. Then, OR chooses a letter $\langle o_j, k^O_j \rangle \in 2^O \times \mathbb{Q}$ and a state q_{j+1} such that there is a transition $\langle q_j, i_j \cup o_j, \gamma_j, q_{j+1} \rangle \in \Delta$ for some

$\gamma_j \in \mathcal{G}_X$, and there exists $f : X \to \mathbb{Q}$ such that $\langle k_t^I, k_t^O \rangle \models_f \gamma_t$ for all $0 \leq t \leq j$. If no such state exists, or if $q_j = \bot$, then OR chooses $q_{j+1} = \bot$. The successive location of the game is q_{j+1}. That is, in every round, every player chooses in its turn a letter, and Player OR chooses a transition that respects all the choices made so far. If no such transition exists, then the game moves to the location $\bot$. If q_{j+1} is $\bot$, then AND wins. Otherwise, the game continues forever and OR wins. Indeed, then, the word $\langle \sigma_0, k_0^I, k_0^O \rangle, \langle \sigma_1, k_1^I, k_1^O \rangle, \cdots \in (2^{I \cup O} \times \mathbb{Q} \times \mathbb{Q})^\omega$ that AND and OR generate during an infinite play is accepted by $\mathcal{A}$.

Note that a position of $G^{\mathcal{A}}$ includes more information than its location. Namely, it has to maintain the guards on the transitions and the numbers that were chosen by Players AND and OR during previous rounds. However, this amounts an unbounded information. Thus, a graph that describes the positions of $G^{\mathcal{A}}$ has an infinite state space. Below we show that for the case of NLWAs^{IO} with simple guards, this graph can be represented symbolically. Essentially, rather than maintaining the values accumulated so far, the graph only maintains restrictions they induce.

Before we solve the game, we examine its usefulness.

Lemma 1. *There is a realizable DPA-NLWA^{IO} $\mathcal{A}$, such that* OR *does not win the synthesis game on $G^{\mathcal{A}}$.*

Proof. Consider the DPA-NLWA $\mathcal{A}$ in Figure 3, over a single variable x and $I = \{i\}$. It is easy to see that $\mathcal{A}$ is DPA-NLWA and that it accepts all words in $(2^I \times \mathbb{Q})^\omega$. Indeed, the run $q_0 \cdot q_1 \cdot {q_3}^\omega$ is an accepting run on words in which the second letter is i, and the run $q_0 \cdot q_2 \cdot {q_4}^\omega$ is an accepting run on words in which the second letter is i. Hence, $\mathcal{A}$ is realizable. However, in the synthesis game, choosing a transition that leaves q_0 amounts to guessing whether the next input is i or $\neg i$, thus OR looses the synthesis game $G^{\mathcal{A}}$. □

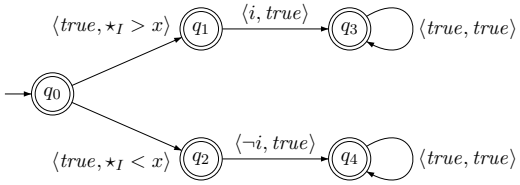


Fig. 3. A realizable DPA-NLWA^{IO} for which OR looses the synthesis game.

Lemma 2. *For every SD-NLWA^{IO} $\mathcal{A}$, we have that* OR *wins $G^{\mathcal{A}}$ iff $\mathcal{A}$ is realizable.*

Proof. Consider an SD-NLWA^{IO} $\mathcal{A}$. It is easy to see that a winning strategy for OR in $G^{\mathcal{A}}$ induces a transducer that realizes $\mathcal{A}$. We prove the second direction. Assume that $\mathcal{A}$ is realizable. Thus, there is a function $f : (2^I \times \mathbb{Q})^* \to (2^O \times \mathbb{Q})$ such that for every $w_I = \langle i_0, k_0^I \rangle \cdot \langle i_1, k_1^I \rangle \cdots \in (2^I \times \mathbb{Q})^\omega$, we have that $w_I \oplus f(w_I) \in L(\mathcal{A})$, where $f(w_I) = f(\langle i_0, k_0^I \rangle) \cdot f(\langle i_0, k_0^I \rangle \cdot \langle i_1, k_1^I \rangle) \cdots \in (2^O \times \mathbb{Q})^\omega$ and $w_I \oplus f(w_I)$ is the infinite word over $2^{I \cup O} \times \mathbb{Q} \times \mathbb{Q}$ combined from w_I and $f(w_I)$. We describe a winning strategy for OR in $G^{\mathcal{A}}$. For $j \geq 0$, let $w_I^j = \langle i_1, k_1^I \rangle, \ldots, \langle i_j, k_j^I \rangle$ be the sequence of letters that AND chose until the j-th round. In the j-th round, if the location of the game is q, then a winning strategy for OR is to move to the only state q' such that

$q' \in \delta(q, w_I^j \oplus f(w_I^j))$. Since $\mathcal{A}$ is semantically deterministic, there is exactly one such q', and by the definition of f, this strategy is winning for OR. □

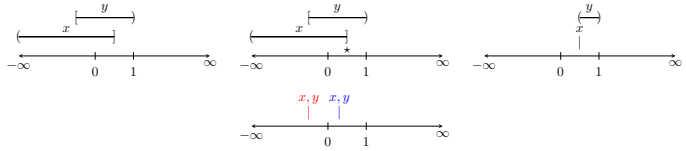
4.2 Solving the synthesis game

In this section we solve the synthesis game for SD-NLWAs^{IO} with simple guards. As stated in Lemma 2, semantic determinization guarantees that the game captures the synthesis problem, and we are left with the challenge of handling the infinite state space. We first need some definitions and notations.

For a set of variables X, we say that a set of brackets $B_X \subset \{\#_x : x \in X, \# \in \{(,), [,]\}\}$ is *legal* if it includes exactly one right and one left bracket for every variable. For example, for $X = \{x, y\}$, the set $\{(_x,]_x, [_y,]_y\}$ is legal, and the set $\{(_x,]_x, [_y\}$ is not. An *interval set* is $N \cup \{-\infty, \infty\} \cup B_X$, where $N \subset \mathbb{Q}$ is a set of numbers and B_X is a legal set of brackets. Consider an interval set $\mathcal{I} = N \cup \{-\infty, \infty\} \cup B_X$, and let $\preceq$ be a total order relation on the elements of $\mathcal{I}$. We say that $\langle \mathcal{I}, \preceq \rangle$ is an *interval description* (ID, for short) if the following hold.

- $k_1 \preceq k_2$ iff $k_1 \leq k_2$ for all $k_1, k_2 \in N \cup \{-\infty, \infty\}$,
- $-\infty \preceq b$ and $\infty \succeq b$ for all $b \in B_X$. If $b \preceq -\infty$, then $b \in \{(_x : x \in X\}$, and if $b \succeq \infty$, then $b \in \{)_x : x \in X\}$,
- $b_1 \preceq b_2$ for all $x \in X$ and $b_1 \in \{(_x, [_x\}, b_2 \in \{)_x,]_x\}$. If $b_2 \preceq b_1$, then $b_1 = [_x$ and $b_2 =]_x$.

As an example, consider $\mathcal{I} = \{0, 1\} \cup \{-\infty, \infty\} \cup \{(_x,]_x, [_y,)_y\}$, and an order relation $\preceq$ such that $-\infty = (_x \prec [_y \prec 0 \prec]_x \prec 1 =)_y \prec \infty$. Then, $\langle \mathcal{I}, \preceq \rangle$ is the ID illustrated in Figure 4a. Intuitively, an ID describes intervals of possible values for variables in X, relatively to each other and to the numbers in N. The tuple $\langle \mathcal{I}, \preceq \rangle$ indicates that $-\infty < x \leq k_1$ for some $0 < k_1 < 1$ and that $k_2 \leq y < 1$, for some $k_2 < 0$.



(a) (b) (c) (d)

Fig. 4. Example of an interval description, a positioning description, and their updates.

A *positioning description* (PD, for short) is a pair $\langle \mathcal{I}_\star, \preceq \rangle$, where $\mathcal{I}_\star = \mathcal{I} \cup \{\star\}$, and $\preceq$ is a total order relation on $\mathcal{I}_\star$ that satisfies all the conditions as for IDs, and in addition, $\star \succ -\infty$ and $\star \prec \infty$. Let $\preceq'$ be the order relation obtained from $\preceq$ by reducing $\preceq$ to the elements in $\mathcal{I}$. We then say that $\langle \mathcal{I}_\star, \preceq \rangle$ is obtained from $\langle \mathcal{I}, \preceq' \rangle$. Note that different PDs can be obtained from a single ID. Intuitively, a PD abstracts a choice of a number by denoting it with $\star$ and describing its position with respect to the intervals of possible values for the variables.

Given a PD $\langle \mathcal{I}_\star, \preceq \rangle$ and a simple guard $\gamma \in \mathcal{G}_X$, one can compute the set $update(\langle \mathcal{I}_\star, \preceq \rangle, \gamma)$ of all IDs that combine the restrictions in γ with the interval-restrictions in $\langle \mathcal{I}_\star, \preceq \rangle$. As an example, consider the PD $\langle \mathcal{I}_\star, \preceq \rangle$ that is illustrated in Figure 4b, namely

$-\infty = ({}_x \prec [{}_y \prec 0 \prec]_x = \star \prec 1 =)_y \prec \infty$. Figure 4c illustrates the (single) ID in $update(\langle \mathcal{I}_\star, \preceq \rangle, (\star = x) \wedge (\star < y))$, namely $-\infty \prec 0 \prec [_x =]_x = ({}_y \prec 1 =)_y \prec \infty$. Figure 4d illustrates, in red and in blue, two IDs in $update(\langle \mathcal{I}_\star, \preceq \rangle, (x = y))$. Note that the guard may contradict the restrictions in the positioning vector. For example, $update(\langle \mathcal{I}_\star, \preceq \rangle, (\star < x)) = \emptyset$.

Note that we define the updating only for simple guards, that is, ones in which each term refers to at most one member in $X \cup \{\star\}$. For example, we do not aim to express the restrictions imposed by $\langle \mathcal{I}_\star, \preceq \rangle$ and the guard $\star \geq x - y + 1$ via an intervals vector. In addition, we point to the following. First, as we demonstrated above, there might be several ways to combine restrictions in a positioning vector with a guard. This multiplicity may derive either from the fact that the guard does not refer to $\star$ (as in the example above), or from disjunctions in the guard. Second, note that one may handle atomic guards of the form $\star \sim x + m$ and $y \sim x + m$, for $\sim \in \{\leq, \geq, =, <, >\}$, $x, y \in X$, and $m \in \mathbb{Q}$, by adding $\{\#_{x+m} : \# \in \{(,), [,]\}\}$ to the brackets set. This requires a prior knowledge of which atomic guards we may have to handle. However, as we show below, when we solve synthesis, the information about possible guards is available.

Theorem 5. *Let $\mathcal{A}$ be a simple NLWA^{IO} over a set X of variables with a set Q of states. The synthesis game $G^{\mathcal{A}}$ is solvable in time polynomial in $|Q|$ and exponential in $|X|$.*

Proof. Given an NLWA^{IO} $\mathcal{A}$, we model $G^{\mathcal{A}}$ by an and-or graph $\langle V_{\text{AND}}, V_{\text{OR}}, E \rangle$. Every vertex in this graph abstracts a position of $G^{\mathcal{A}}$, indicating whose is the current turn of the play, the location of the game, the letters and numbers AND and OR choose in the current round, and OR's commitments regarding the values of the variables, induced by previous transitions he chose. These commitments are expressed via IDs, and the $\mathbb{Q}$-choices of the players are expressed via PDs. We maintain a single PD with two $\star$-s, abbreviated with I and O to indicate whose choice they are. We extend the notion of PDs accordingly, so that they include both $\star_I$ and $\star_O$.

Formally, let $\mathcal{A}$ be an NLWA^{IO} over a set of variables X, and let $N \subset \mathbb{Q}$ be the set of constants that appear in the guards of $\mathcal{A}$. We denote by $\hat{\mathcal{I}}$ the set of all interval sets $N \cup \{-\infty, \infty\} \cup B_X$ for legal B_X, and define $P = \{\langle \mathcal{I}, \preceq \rangle : \mathcal{I} \in \hat{\mathcal{I}}$ and $\langle \mathcal{I}, \preceq \rangle$ is an ID$\}$. In addition, let $P_{\star_I}$ and $P_{\star_I, \star_O}$ be the set of all PDs obtained from IDs in P by adding to them the symbols $\star_I$ and $\star_I, \star_O$, respectively.

We define $V_{\text{AND}} = Q \times P$ and $V_{\text{OR}} = (Q \times 2^I \times P_{\star_I}) \cup (Q \times 2^{I \cup O} \times P_{\star_I, \star_O})$. Let $V = V_{\text{AND}} \cup V_{\text{OR}}$. Then, $E \subseteq V \times V$, is the transition relation that consists of the following transitions, for every $q \in Q$, $i \in 2^I$ and $o \in 2^O$:

- $(\langle q, \langle \mathcal{I}, \preceq \rangle \rangle, \langle q, i, \langle \mathcal{I}_{\star_I}, \preceq' \rangle \rangle)$, where $\langle \mathcal{I}_{\star_I}, \preceq' \rangle \in P_{\star_I}$ is a PD obtained from the ID $\langle \mathcal{I}, \preceq \rangle \in P$,
- $(\langle q, i, \langle \mathcal{I}_{\star_I}, \preceq \rangle \rangle, \langle q, i \cup o, \langle \mathcal{I}_{\star_I, \star_O}, \preceq' \rangle \rangle)$, where $\langle \mathcal{I}_{\star_I, \star_O}, \preceq' \rangle \in P_{\star_I, \star_O}$ is a PD obtained from the PD $\langle \mathcal{I}_{\star_I}, \preceq \rangle \in P_{\star_I}$, and
- $(\langle q, i \cup o, \langle \mathcal{I}_{\star_I, \star_O}, \preceq \rangle \rangle, \langle q', \langle \mathcal{I}, \preceq' \rangle \rangle)$, where there is a transition $\langle q, i \cup o, \gamma, q' \rangle$ in $\mathcal{A}$, and $\langle \mathcal{I}, \preceq' \rangle \in update(\langle \mathcal{I}_{\star_I, \star_O}, \preceq \rangle, \gamma)$.

Note that vertices in $Q \times 2^{I \cup O} \times P_{\star_I, \star_O}$ might not have a successor, for example, if q does not have an $(i \cup o)$-successor, or if $update(\langle \mathcal{I}_{\star_I, \star_O}, \preceq \rangle, \gamma) = \emptyset$.

Finally, we define $v_0 = \langle q_0, \langle \mathcal{I}^0, \preceq_0 \rangle \rangle$, where $\mathcal{I}^0 = N \cup \{-\infty, \infty\} \cup \{(_x\colon x \in X\} \cup \{)_x : x \in X\}$, and $\preceq_0$ is the only possible order relation on the elements in $\mathcal{I}^0$.

The players generate a play over $\langle V_{\text{AND}}, V_{\text{OR}}, E \rangle$ as follows. In every round, if the current position is $v \in V_j$, for $j \in \{\text{AND}, \text{OR}\}$, then player j chooses a successor v' of v, and the play proceeds to position v'. Recall that $V = V_{\text{OR}} \cup V_{\text{AND}}$. A *strategy* for a player $j \in \{\text{AND}, \text{OR}\}$ is a function $f_j : V^* \times V_j \to V$ such that for every $u \in V^*$ and $v \in V_j$, we have that $\langle v, f_j(u, v) \rangle \in E$. Thus, a strategy for Player j maps the history of the game so far, when it ends in a position v owned by player j, to a successor of v. Two strategies $f_{\text{AND}}, f_{\text{OR}}$ and the initial position v_0 induce a play $\pi = v_0, v_1, v_2 \cdots \in V^\omega$, where for every $i \geq 0$, if $v_i \in V_j$ for $j \in \{\text{AND}, \text{OR}\}$, then $v_{i+1} = f_j((v_0, \ldots, v_{i-1}), v_i)$. We say that π is the *outcome* of $f_{\text{OR}}, f_{\text{AND}}$ and v_0, and denote $\pi = \mathit{outcome}(f_{\text{OR}}, f_{\text{AND}}, v_0)$. A play π is *winning* for OR if it is infinite. A position $v \in V$ is winning for OR if there exists a strategy f_{OR} such that for every strategy f_{AND}, we have that $\mathit{outcome}(f_{\text{OR}}, f_{\text{AND}}, v)$ is winning for OR. Then, deciding who wins $G^{\mathcal{A}}$ reduces to deciding whether the vertex that abstracts the initial position of the game is winning for AND.

It is not hard to see that the game $\langle V_{\text{AND}}, V_{\text{OR}}, E \rangle$ models the synthesis game. Note that a winning strategy for OR in this graph induces a transducer that realizes $L(\mathcal{A})$. Indeed, a PD essentially abstracts the $\mathbb{Q}$-choices of the environment and the system. The former can be described using guards over X and $\star_I$ that label the transitions that leave the according state in the transducer. The later induces a guard over X and $\star_O$ that labels the according state in the transducer. Finally, since the graph is finite and every infinite play is winning, the problem of deciding whether v_0 is winning for Player OR reduces to the problem of reachability in and-or graphs, which can be solved in time polynomial in the size of the graph. It is not hard to see that the size of the graph is polynomial in $|Q|$ and in $|P \cup P_{\star_I} \cup P_{\star_I, \star_O}|$, and that the later is exponential in $|X|$. Therefore, one can decide who wins $G^{\mathcal{A}}$ in time polynomial in $|Q|$ and exponential in $|X|$. □

Together with Lemma 2, Theorem 5 implies the following.

Corollary 1. *The synthesis problem for a simple SD-NLWA^{IO} over a set X of variables with a set Q of states can be is solved in time polynomial in $|Q|$ and exponential in $|X|$.*

Note that while for model checking, a framework that handles $\mathbb{Q}$ is more general than one that handles $\mathbb{N}$, for synthesis this is not the case. That is, there are specifications that are realizable over $\mathbb{Q}$, but not over $\mathbb{N}$. For example, a specification in which the system has to choose a number between two numbers given by the environment. In particular, the abstraction in our synthesis algorithm exploits the density of $\mathbb{Q}$. We leave the question of solving synthesis for NLWAs over $\Sigma \times \mathbb{N}$ open.

References

1. R. Bloem, K. Chatterjee, and B. Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking.*, pages 921–962. Springer, 2018.
2. M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and xml reasoning. *Journal of the ACM*, 56(3):1–48, 2009.

3. A. Bouajjani, P. Habermehl, and R R. Mayr. Automatic verification of recursive procedures with one integer parameter. *TCS*, 295:85–106, 2003.
4. M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescu. Specification and design of workflow-driven hypertexts. *J. Web Eng.*, 1(2):163–182, 2003.
5. S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., 2002.
6. A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.
7. G. Delzanno, A. Sangnier, and R. Traverso. Parameterized verification of broadcast networks of register automata. In *Reachability Problems*, pages 109–121, Springer, 2013.
8. R. Ehlers, S. Seshia, and H. Kress-Gazit. Synthesis with identifiers. In *Proc. 15th VMCAI*, LNCS 8318, pages 415–433. Springer, 2014.
9. L. Exibard, E. Filiot, and P-A. Reynier. Synthesis of data word transducers. In *Proc. 30th CONCUR*, 2019.
10. R. Faran and O. Kupferman. LTL with arithmetic and its applications in reasoning about hierarchical systems. In *Proc. 22nd LPAR*, EPiC 57, pages 343–362, 2018.
11. O. Grumberg, O. Kupferman, and S. Sheinvald. An automata-theoretic approach to reasoning about parameterized systems and specifications. In *11th ATVA*, pages 397–411, 2013.
12. T.A. Henzinger and N. Piterman. Solving games without determinization. In *Proc. 15th CSL*, LNCS 4207, pages 394–410. Springer, 2006.
13. A. Khalimov and O. Kupferman. Register bounded synthesis. In *Proc. 30th CONCUR*, 2019.
14. A. Khalimov, B. Maderbacher, and R. Bloem. Bounded synthesis of register transducers. In *Proc. 16th ATVA*, LNCS 11138, pages 494–510. Springer, 2018.
15. O. Kupferman, S. Safra, and M.Y. Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006.
16. O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th FoCS*, pages 531–540, 2005.
17. R. Lazić and D. Nowak. A unifying approach to data-independence. In *Proc. 11th CONCUR*, pages 581–596. Springer Berlin Heidelberg, 2000.
18. F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In *26th MFCS*, pages 560–572. Springer-Verlag, 2001.
19. D. Niwinski and I. Walukiewicz. Relating hierarchies of word and tree automata. In *Proc. 15th TACAS*, LNCS 1373. Springer, 1998.
20. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.
21. S. Safra. On the complexity of ω-automata. In *Proc. 29th FoCS*, pages 319–327, 1988.
22. A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.
23. Y. Shemesh and N.: Francez. Finite-state unification automata and relational languages. *Information and Computation*, 114:192–213, 1994.
24. V. Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT '09*, pages 1–13, 2009.

A Proofs

A.1 Proof of Theorem 1

We describe a translation of a given NLWA to an equivalent DPA-NLWA. Consider an NLWA $\mathcal{A} = \langle \Sigma, X, Q, Q_0, \Delta \rangle$. Let $Q = \{q_1, \ldots, q_n\}$. If there is a transition

$\langle q_i, \sigma, \gamma, q_j \rangle \in \Delta$, then we denote γ by γ^σ_{ij}. That is, γ^σ_{ij} is the guard that labels the σ-transition from q_i to q_j, if exists. Note that since guards can be disjunctioned, we can assume that for every two states $q_1, q_2 \in Q$ and a letter $\sigma \in \Sigma$, there is at most one transition $\langle q_1, \sigma, \gamma, q_2 \rangle \in \Delta$. We define $\mathcal{A}' = \langle \Sigma, X', S, s_0, \Delta' \rangle$ as follows. The set of variables X' is $\{x_{ij\sigma} : q_i, q_j \in Q, \sigma \in \Sigma, \text{ and } x \text{ appears in } \gamma^\sigma_{ij}\}$. In other words, we create a copy of every variable for every transition that it appears on. The state space is $S = \{s : Q \to (2^{X'} \cup \bot)\}$. That is, every state in S represents a function that pairs every state in Q with either a subset of X' or the character $\bot$. We define the initial state s_0 such that $s_0(q) = \emptyset$ for every $q \in Q_0$, and $s_0(q) = \bot$ for every $q \in Q \setminus Q_0$.

Finally, we define $\Delta' \subseteq S \times \Sigma \times \mathcal{G}_{X'} \times S$. We start with several notations. For every state $s \in S$, let us denote $states(s) = \{q : s(q) \neq \bot\}$. For every guard γ^σ_{ij}, we denote by $\hat{\gamma}^\sigma_{ij}$ the guard obtained from γ^σ_{ij} by replacing every variable x that appears in γ^σ_{ij} by $x_{ij\sigma}$. For a set of variables $Y \subseteq X'$, we denote by $\gamma_=(Y)$ the guard that forces all the variables in Y that are obtained from the same variable in X to have the same value. For example, if $Y = \{x_{i_1 j_1 \sigma_1}, x_{i_2 j_2 \sigma_2}, y_{i_3 j_3 \sigma_3}\}$, then $\gamma_=(Y)$ is $x_{i_1 j_1 \sigma_1} = x_{i_2 j_2 \sigma_2}$. For a state $s \in S$, we denote $\gamma^s_= = \bigwedge_{q \in states(s)} \gamma_=(s(q))$. That is, $\gamma^s_=$ forces all the variables in $s(q)$ that are obtained from the same variable in X to have the same value, for every $q \in states(s)$. Finally, for a state $s \in S$ and a letter $\sigma \in \Sigma$, we denote $succ_\sigma(s) = \{q \in Q : \text{ there is a } \sigma\text{-transition from a state in } states(s) \text{ to } q\}$.

The transition relation Δ' includes, for every state s_1, letter $\sigma \in \Sigma$, and nonempty subset $T \subseteq succ_\sigma(s_1)$, the transition $\langle s_1, \sigma, \bar{\gamma}, s_2 \rangle$, where $states(s_2) = T$, and for every j such that $q_j \in T$ we have that $s_2(q_j) = \bigcup \{s_1(q_k) \cup X'_{kj\sigma} : k \text{ is such that } q_k \in states(s_1) \text{ and } \langle q_k, \sigma, \gamma^\sigma_{kj}, q_j \rangle \in \Delta\}$, and $X'_{kj\sigma} = \{x' \in X' : x' \text{ appears in } \hat{\gamma}^\sigma_{kj}\}$. In addition, $\bar{\gamma} = \gamma^{s_2}_= \wedge \bigwedge_{i:q_i \in states(s_1)} (\bigwedge_{j:q_j \in T} \hat{\gamma}^\sigma_{ij} \wedge \bigwedge_{j:q_j \in succ(s_1) \setminus T} \neg \hat{\gamma}^\sigma_{ij})$.

We turn to explain the transition relation of $\mathcal{A}'$. First, there is a σ-transition from $s_1 \in S$ to $s_2 \in S$ iff $states(s_2)$ is a nonempty subset of the σ-successors of $states(s_1)$. For every such subset, we construct a σ-transition in Δ' that stands for the case that the guards that are satisfied are exactly the ones that label the transitions that enter this subset. Let $\mathcal{G}$ be the set of those guards. Then, the corresponding transition is labeled with a guard that conjuncts all the guards in $\mathcal{G}$ and all the negations of guards not in $\mathcal{G}$. Finally, every state in $states(s_2)$ is paired with the variables that appear on previous transitions. As an example, consider a transition $\langle q_1, \sigma, \gamma^\sigma_{12}, q_2 \rangle \in \Delta$, and a state s_1 such that $states(s_1) = \{q_1\}$. The set $T = \{q_2\}$ induces a transition $\langle s_1, \sigma, \bar{\gamma}, s_2 \rangle \in \Delta'$, where $\bar{\gamma} = \gamma^{s_2}_= \wedge \hat{\gamma}^\sigma_{12} \wedge \bigwedge_{j:q_j \in succ_\sigma(s_1) \setminus \{q_2\}} \neg \hat{\gamma}^\sigma_{1j}$, $states(s_2) = \{q_2\}$, and $s_2(q_2) = s_1(q_1) \cup X'_{12\sigma}$. In other words, q_2 is paired with all the variables that q_1 is paired with, and in addition, all the variables that appear in $\hat{\gamma}^\sigma_{12}$. Intuitively, this indicates that all the variables in $s_2(q_2)$ that are obtained from the same variable in X are occurrences of this variable that should not be distinguished. Finally, note that whenever we move to a state s in which some variables should not be distinguished, the guard on the transition that enters s forces these variables to have the same value.

We prove that $L(\mathcal{A}') = L(\mathcal{A})$. For $f : X \to \mathbb{Q}$, we denote $f' : X' \to \mathbb{Q}$ such that $f'(x_{ij\sigma}) = f(x)$ for every $1 \leq i, j \leq n$ and $\sigma \in \Sigma$. We show that for every word $w \in (\Sigma \times \mathbb{Q})^*$ and every assignment $f : X \to \mathbb{Q}$, there is a run r of $\mathcal{A}$ on w under f that reaches q iff the run of $\mathcal{A}'$ on w under f' reaches a state s such that $q \in states(s)$, and $s(q)$ is such that if r includes the transition $\langle q_i, \sigma, \gamma, q_j \rangle$ and $x \in X$ appears in γ

then $x_{ij\sigma} \in s(q)$. It is easy to see that the claim follows. Indeed, we have that for every word $w \in (\Sigma \times \mathbb{Q})^\omega$ and assignment function $f : X \to \mathbb{Q}$, there is a run of $\mathcal{A}$ on w under f iff there is a run of $\mathcal{A}'$ on w under f'.

The proof proceeds by induction on the length of w. For the empty word, the claim is trivial. Now, assume that the claim holds for every word of length at most k, for some $k \geq 0$. Let $w = \langle \sigma_1, d_1 \rangle \dots \langle \sigma_{k+1}, d_{k+1} \rangle$ be a word of length $k+1$, and let $f : X \to \mathbb{Q}$ be an assignment function. Let $s^i \in S$ be the state that $\mathcal{A}'$ reaches after reading $\langle \sigma_1, d_1 \rangle \dots \langle \sigma_i, d_i \rangle$. On one direction, assume that there is a run $r = q_0, q_1, \dots, q_{k+1}$ of $\mathcal{A}$ on w under f. By the induction hypothesis, $q_k \in states(s^k)$ and $s^k(q_k)$ includes all the variables in guards $\hat{\gamma}_{i,i+1}^{\sigma_{i+1}}$ such that there is a transition $\langle q_i, \sigma_{i+1}, \gamma_{i,i+1}^{\sigma_{i+1}}, q_{i+1} \rangle \in \Delta$ and $0 \leq i \leq k-1$. Consider the transition $\langle q_k, \sigma_{k+1}, \gamma_{k,k+1}^{\sigma_{k+1}}, q_{k+1} \rangle \in \Delta$, and the set $T \subseteq succ_{\sigma_{k+1}}(s^k)$ such that d_{k+1} satisfies exactly all the guards on the transitions to T under f. By the definition of Δ', there is a transition $\langle s^k, \sigma_{k+1}, \bar{\gamma}, s^{k+1} \rangle$, where $q_{k+1} \in states(s^{k+1})$, and $s^{k+1}(q_{k+1})$ includes all the variables in $s^k(q_k)$ and the variables that appear in $\hat{\gamma}_{k,k+1}^{\sigma_{k+1}}$. Therefore, $s^{k+1}(q_{k+1})$ includes all the variables that appear on transitions of r. We left to show that $d_{k+1} \models_{f'} \bar{\gamma}$. By the definition of T and f', we have that $d_{k+1} \models_{f'} \bigwedge_{i:q_i \in states(s^k)} (\bigwedge_{j:q_j \in T} \hat{\gamma}_{ij}^{\sigma} \wedge \bigwedge_{j:q_j \in succ_\sigma(s^k) \setminus T} \neg \hat{\gamma}_{ij}^{\sigma})$. As for $\gamma_{=}^{s^{k+1}}$, note that its satisfaction does not depend on d_{k+1}, and that it trivially follows from the definition of f'.

The other direction is easy. Consider the transition $\langle s^k, \sigma_{k+1}, \bar{\gamma}, s^{k+1} \rangle$, and let γ be the guard that labels a σ_{k+1}-transition from a state $q_1 \in states(s^k)$ to a state $q_2 \in states(s^{k+1})$, thus $\langle q_1, \sigma_{k+1}, \gamma, q_2 \rangle \in \Delta$. Note that $d_{k+1} \models_{f'} \gamma$. By the induction hypothesis, there is a run r of $\mathcal{A}$ on $\langle \sigma_1, d_1 \rangle \dots \langle \sigma_k, d_k \rangle$ under f that reaches q_1. Therefore, the run $r \cdot q_2$ is a run of $\mathcal{A}$ on w under f that reaches a state in $states(s^{k+1})$. The claim regarding the variables in $s^{k+1}(q_2)$ follows directly from the construction.

A.2 Proof of Theorem 2

Assume by way of contradiction that there is an SD-NLWA $\mathcal{A}'$ that accepts $L(\mathcal{A})$. We construct two prefixes $w_1, w_2 \in (\Sigma \times \mathbb{Q})^*$ that differ only in their last letter, such that the runs r_1, r_2 of $\mathcal{A}'$ on them reach the same state, with $\gamma^{r_1} = \gamma^{r_2}$, and there is a suffix $x \in (\Sigma \times \mathbb{Q})^\omega$ such that $w_1 \cdot x \in L(\mathcal{A})$ and $w_2 \cdot x \notin L(\mathcal{A})$, contradicting the assumption that $\mathcal{A}'$ is semantically deterministic.

We construct w_1, w_2 iteratively. In every iteration except for the last one, we concatenate the same letter to both w_1 and w_2. In the last iteration, we pick two different letters that lead us to the same state, and both of which do not extend the set of constrains of the run. By the assumption that $\mathcal{A}'$ is SD-NLWA, we have that there is at most one run on every word in $(\Sigma \times \mathbb{Q})^\omega$. For a prefix $w \in (\Sigma \times \mathbb{Q})^*$, we denote by $reach(w)$ the state that the run on w reaches. We now describe how an iteration proceeds. Let $w = u_0 \cdot \dots \cdot u_n$ be the common prefix constructed until the current iteration. For an index $i \geq 0$, let $w[i] = u_0 \dots u_i$. We say that an iteration is *winning* if there are two numbers $k_1, k_2 \in \mathbb{Q}$ and a state q', such that $k_1 \neq k_2$, $reach(w \cdot \langle a, k_1 \rangle) = reach(w \cdot \langle a, k_2 \rangle) = q'$, and there are two indices $0 \leq i, j < n$ such that $reach(w) = reach(w[i]) = reach(w[j])$, $u_{i+1} = \langle a, k_1 \rangle$, $u_{j+1} = \langle a, k_2 \rangle$, and $q' = reach(w[i+1]) = reach(w[j+1])$. In other words, an iteration is winning

if there are two numbers on which the run proceeds to the same state, and both of the numbers were chosen in previous iterations, where the state was $reach(w)$ and the next state, to which the run proceeded, was q'.

In every iteration, we first check whether it is a winning iteration. If it is, and the two numbers that witness it are k_1 and k_2, then we define $w_1 = w \cdot \langle a, k_1 \rangle$ and $w_2 = w \cdot \langle a, k_2 \rangle$. It is easy to see that the runs of $\mathcal{A}'$ on w_1 and w_2 reach the same state with the same set of constraints. In addition, for the suffix $x = \langle a, k_1 \rangle \cdot \langle b, 0 \rangle^\omega$, we have that $w_1 \cdot x \in L(\mathcal{A})$ and $w_2 \cdot x \notin L(\mathcal{A})$. Therefore, if an iteration is winning, then we are done. A non-winning iteration proceeds as follows. We choose a number $0 \leq k \leq |Q|$, where Q is the set of states of $\mathcal{A}'$, such that there is no prefix $w[i]$ of w for which $reach(w[i]) = reach(w)$, $u_{i+1} = \langle a, k \rangle$, and $reach(w[i+1]) = reach(w \cdot \langle a, k \rangle)$. Then, we concatenate $\langle a, k \rangle$ to w, and proceed to the next iteration. We show that for non-winning iteration, such a number always exists. Since we consider only numbers between 0 and $|Q|$, there are at least two numbers k_1, k_2 such that $reach(w \cdot \langle a, k_1 \rangle) = reach(w \cdot \langle a, k_2 \rangle)$. Since the iteration is not winning, it follows that at least for one of them satisfies the conditions. Finally, note that since we consider only numbers from 0 to $|Q|$ and since the automaton is finite, we eventually reach a winning iteration.

It is interesting to note that the proof is independent of the acceptance condition, thus there is no semantically deterministic automaton equivalent to $\mathcal{A}$ even if we use richer types of acceptance conditions.