Variable Automata over Infinite Alphabets

Orna Grumberg[a], Orna Kupferman[b], Sarai Sheinvald[b]

[a] *Department of Computer Science, The Technion, Haifa 32000, Israel*
[b] *School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel*

Abstract

Automated reasoning about systems with infinite domains requires an extension of automata, and in particular, regular automata, to *infinite alphabets*. Existing formalisms of such automata cope with the infiniteness of the alphabet by adding to the automaton a set of registers or pebbles, or by attributing the alphabet by labels from an auxiliary finite alphabet that is read by an intermediate transducer. These formalisms involve a complicated mechanism on top of the transition function of automata over finite alphabets and are therefore difficult to understand and to work with.

We introduce and study *variable finite automata over infinite alphabets* (VFA). VFA form a natural and simple extension of regular (and ω-regular) automata, in which the alphabet consists of letters as well as variables that range over the infinite alphabet domain. Thus, VFAs have the same structure as regular automata, only that some of the transitions are labeled by variables. We compare VFA with existing formalisms, and study their closure properties and classical decision problems. We consider the settings of both finite and infinite words. In addition, we identify and study the deterministic fragment of VFA. We show that while this fragment is sufficiently strong to express many interesting properties, it is closed under union, intersection, and complementation, and its nonemptiness and containment problems are decidable. Finally, we describe a determinization process for a determinizable subset of VFA.

Keywords: Automata, infinite alphabets, verification, model checking

1. Introduction

Automata-based formal methods are successfully applied in automated reasoning about systems. When the systems are finite-state, their behaviors and specifications can be modeled by finite automata. When the systems are infinite-state, reasoning is undecidable, and research is focused on identifying decidable

Email addresses: orna@cs.technion.ac.il (Orna Grumberg), orna@cs.huji.ac.il (Orna Kupferman), surke@cs.huji.ac.il (Sarai Sheinvald)

special cases (e.g., pushdown systems) and on developing heuristics (e.g., abstraction) for coping with the general case.

One type of infinite-state systems, motivating this work, are systems in which the control is finite and the source of infinity is data. This includes, for example, software with integer parameters [3], datalog systems with infinite data domain [16, 4], and XML documents, whose leaves are typically associated with data values from some infinite domain [7, 5]. Lifting automata-based methods to the setting of such systems requires the introduction of automata with *infinite alphabets*. [1]

The transition function of a nondeterministic automaton over finite alphabets (NFA) maps a state q and a letter σ to a set of states the automaton may move to when it is in state q and the letter in the input is σ. When the alphabet of the automaton is infinite, specifying all transitions is impossible, and a new formalism is needed in order to represent them in a finite manner. Existing formalisms of automata with infinite alphabets fulfill this task by augmenting the automaton by *registers* or *pebbles*, or by attributing the alphabet by labels from an auxilary finite alphabet that is read by an intermediate transducer.

The quality of a formalism is measured by its simplicity, expressive power, compositionality, and computability. In *simplicity*, we refer to the effort required in order to understand a given automaton, work with it, and implement it. In *compositionality*, we refer to closure under the basic operations of union, intersection, and complementation. In *computability*, we refer to the decidability and complexity of classical problems like nonemptiness, membership, universality, and containment.

In this work [2], we introduce and study a new formalism for recognizing languages over infinite alphabets. Our formalism, *variable finite automata* (VFA), forms a natural and simple extension of NFAs. We also identify and study a fragment of VFA that fulfills the simplicity, compositionality, and computability criteria, and is still sufficiently expressive to specify many interesting properties. Intuitively, a VFA is an NFA some of whose letters are variables ranging over the infinite alphabet. The tight connection with NFAs enables us to apply much of the constructions and algorithms known for them.

Before elaborating on our formalism, we survey the existing formalisms for automata over infinite alphabets we have mentioned.

A register automaton [14] has a finite set of registers, each of which may contain a letter from the infinite alphabet. The transitions of a register automaton 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, [11] forces the content of the registers to be different, [13] adds alternation and two-wayness, and [10] allows the registers to change their content nondeterministically during the run.

[1] Different approaches for automatically reasoning about such systems are based on extensions of first-order logic [2] and linear temporal logics [8].

[2] This work is an extended version of our paper [9]

A pebble automaton [13] places pebbles on the input word in a stack-like manner. The transitions of a pebble automaton compare the letter in the input with the letters in positions marked by the pebbles. Several variants of this model have been studied. For example, [13] studies alternating and two-way pebble automata, and [15] introduces top-view weak pebble automata.

The newest formalism is *data automata* [2, 1]. For an infinite alphabet Σ, a data automaton runs on *data words*, which are words over the alphabet $\Sigma \times F$, where F is a finite auxilary alphabet. Intuitively, the finite alphabet is accessed directly, while the infinite alphabet can only be tested for equality, and is used for inducing an equivalence relation on the set of positions. Technically, a data automaton consists of two components. The first is a letter-to-letter transducer that runs on the projection of the input word on F and generates words over yet another alphabet Γ. The second is a regular automaton that runs on subwords (determined by the equivalence classes) of the word generated by the transducer.

The formalisms of register, pebble, and data automata all fail hard the simplicity criterion. Augmenting NFAs with registers or pebbles requires a substantial modification of the transition function. The need to maintain the registers and pebbles makes the automata hard to understand and work with. Unfortunately, most researchers in the formal-method community are not familiar with register and pebble automata. Indeed, even the definition of the basic notion of a run of such automata cannot simply rely on the familiar definition of a run of an NFA, and involves the notions of configurations, successive configurations, and so on, with no possible shortcuts.

Data automata do not come to the rescue. The need to accept several subwords per input word and to go through an intermediate alphabet and transducer makes them very complex. Even trivial languages such as a^* require extra letters and checks in order to be recognized. Simplicity is less crucial in the process of automatic algorithms, and indeed, data automata have been succesfully used for the decidability of two-variable first order logic on words with data - a formalism that is very useful in XML reasoning [2, 1]. For the purpose of specification and design, and for developing new algorithms and applications, simplicity is crucial. A simpler, friendlier formalism is needed.

Data and register automata and most of their variants fail the compositionality and computability criteria too. Data automata and register automata are not closed under complementation, apart from specific fragments of register automata that limit the number of registers [8]. Their universality and containment problems are undecidable [13]. Pebble automata and most of their variants fail the computability criterion, as apart from weaker models [15], their nonemptiness, universality, and containment problems are undecidable. Nonemptiness of data and register automata is decidable, but is far more complex than the easy reachability-based nonemptiness algorithm for NFAs.

We now continue to elaborate on our formalism. Formally, a VFA is a pair $\mathcal{A} = \langle \Sigma, A \rangle$, where Σ is an infinite alphabet and A is an NFA, referred to as the *pattern automaton* of $\mathcal{A}$. The alphabet of A consists of *constant letters* – a finite subset of Σ, a set of *bounded variables*, and a single *free variable*. The language of $\mathcal{A}$ consists of words in Σ^* that are formed by assigning letters in Σ to the

occurrences of variables in words in the language of A. Each bounded variable is assigned a different letter (also different from the constant letters), thus all occurrences of a particular bounded variable must be assigned the same letter. This allows describing words in Σ^* in which some letter is repeated. The free variable may be assigned different letters in every occurrence, different from the constant letters and from letters assigned to the bounded variables. This allows describing words in which every letter may appear. For example, consider a VFA $\mathcal{A} = \langle \mathbb{N}, A \rangle$, where A has a bounded variable x and its free variable is y. If the language of A is $(x+y)^* \cdot x \cdot (x+y)^* \cdot x \cdot (x+y)^*$, then the language of $\mathcal{A}$ consists of all words over $\mathbb{N}$ in which at least some letter occurs at least twice.

We prove that VFAs are closed under union and intersection. The constructions we present use the union and product constructions for NFAs in their basis, but some pirouettes are needed in order to solve conflicts between different assignments to the variables of the underlying automata. Such pirouettes are helpless for the problem of complementation, and we prove that VFAs are not closed under complementation. We study the classical decision problems for VFAs. We show that a VFA is nonempty iff its pattern automaton is nonempty. Thus, the nonemptiness problem is NL-complete, and is not more complex than the one for NFAs. We also show that the membership problem is NP-complete. Thus, while the problem is more complex than the one for NFAs, it is still decidable. The universality and containment problems, however, are undecidable.

We then define and study *deterministic VFA* (DVFA), a fragment of VFA in which there exists exactly one run on every word. Unlike the case of DFAs, determinism is not a syntactic property. Indeed, since the variables are not preassigned, there may be several runs on a word even when the pattern automaton is deterministic. However, a syntactic definition does exist and deciding whether a given VFA is deterministic is NL-complete. We introduce an *unwinding operator* for VFAs. In an unwound VFA, each state is labeled by the variables that have been read, and therefore assigned, in paths leading to the state. Using the unwinding operator, we can define DVFAs for the union and intersection of DVFAs. Moreover, the closure under complementation of DVFAs is immediate, and it enables us to solve the universality and containment problems for DVFAs. Thus, DVFAs suggest an expressive formalism that also fulfills the criteria of simplicity, computability, compositionality.

We study further properties of DVFA. As bad news, we show that the problem of determinizing a given VFA (or concluding that no equivalent DVFA exists) is undecidable. As good news, we show that all VFAs with no free variable have an equivalent DVFA, and present a determinization procedure for VFAs of this kind. The advantages of DVFA make us optimistic about the extensions of algorithms that involve DFAs, like symbolic formal verification and synthesis, to the setting of infinite alphabets.

We demonstrate the robustness of our formalism by showing that its extension to the setting of ω-regular words is straightforward. In Section 7, we introduce and study *variable Büchi automata* (VBAs), whose pattern automata are nondeterministic Büchi automata on infinite words [6]. VBAs are useful for specifying languages of infinite words over infinite alphabets, and in particular,

specifications of systems with variables ranging over infinite domains. We show that the known relation between NFAs and nondeterministic Büchi automata extends to a relation between VFAs and VBAs. This enables us to easily lift the properties and decision procedures we presented for VFA to the setting of VBAs.

2. Variable Automata over Infinite Alphabets

A nondeterministic finite automaton (NFA) is a tuple $A = \langle \Gamma, Q, Q_0, \delta, F \rangle$, where Γ is a finite alphabet, Q is a finite set of *states*, $Q_0 \subseteq Q$ is a set of *initial states*, $\delta : Q \times \Gamma \to 2^Q$ is a *transition function*, and $F \subseteq Q$ is a set of *accepting states*. If $\delta(q, a) \neq \emptyset$, we say that *a exits q*. A *run of A* on a word $w = \sigma_1 \sigma_2 \dots \sigma_n$ in Γ^* is a sequence of states $r = r_0, r_1, \dots, r_n$ such that $r_0 \in Q_0$ and for every $1 \leq i \leq n$ it holds that $r_i \in \delta(r_{i-1}, \sigma_i)$. If $r_n \in F$ then r is *accepting*. Note that a run may not exist. If a run does exist, we say that w is *read along* A. The language of A, denoted $L(A)$, is the set of words w such that there exists an accepting run of A on w.

Before defining variable automata with infinite alphabets, let us explain the idea behind them. Consider the NFA A_1 over the finite alphabet $\{x, y\}$ appearing in Figure 1. It is easy to see that $L(A_1) = x \cdot y^* \cdot x$. Suppose now that we want to define an automaton for the language $L = \{i \cdot y^* \cdot i : i \in \mathbb{N}\}$ over the infinite alphabet $\{y\} \cup \mathbb{N}$. One naive way to do so is to branch, after reading the first letter, to a state that remembers it. This, however, requires an infinite state space. The way register and pebble automata address this problem is by storing the first letter in a register or placing a pebble on it, and then comparing the letters in the input with the letter stored in the register or the letter on which a pebble is placed. This requires a complicated transition function that involves registers or pebbles and their maintenance. Our variable automata are based on the following simple idea: we stay with the NFA A_1, but rather than refering to x as a letter in the alphabet, we refer to x as a variable that ranges over $\mathbb{N}$.

Next, suppose we want to define an automaton for the language $L' = \{i_1 \cdot i_2 \cdots i_k : k \geq 2, i_j \in \mathbb{N}, i_1 = i_k, \text{ and } i_j \neq i_1 \text{ for all } 1 < j < k\}$ over the alphabet $\mathbb{N}$; that is, L' contains exactly all words in which the first letter is equivalent to the last letter, and is different from all other letters. Again, the straightforward solution involves an infinite state space, and register and pebble automata are complicated. Our solution is to refer to both x and y as variables with range $\mathbb{N}$ as follows: x is a bounded variable whose value is fixed once assigned, and y is a free variable that can take changing values (different from the value assigned to x). This way, A_1 recognizes L'. Also, if we want to remove the restriction about the letters in the middle being different from the first letter, thus consider $L'' = \{i_1 \cdot i_2 \cdots i_k : k \geq 2, i_j \in \mathbb{N}, \text{ and } i_1 = i_k\}$, we can label the self loop in A_1 by both x and y.

We now define *variable finite automata* (VFAs) formally. A VFA is a pair $\mathcal{A} = \langle \Sigma, A \rangle$, where Σ is an infinite alphabet and A is an NFA, to which we refer as the *pattern automaton of* $\mathcal{A}$. The (finite) alphabet of A is $\Gamma_A = \Sigma_A \cup X \cup \{y\}$,

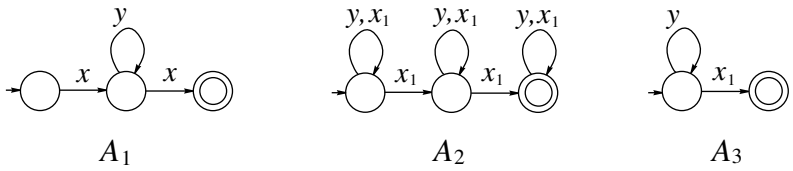

Figure 1: The pattern automata A_1, A_2, and A_3 for the VFAs $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$.

where $\Sigma_A \subset \Sigma$ is a finite set of *constant letters*, X is a finite set of *bounded variables* and y is a *free variable*. We refer to the number of bounded variables in A as the *width of* A. The variables in $X \cup \{y\}$ range over $\Sigma \setminus \Sigma_A$.

Consider a word $v = v_1 v_2 \dots v_n \in \Gamma_A^*$ read along A, and another word $w = w_1 w_2 \dots w_n \in \Sigma^*$. We say that w is a *legal instance of* v *in* $\mathcal{A}$ if

- $v_i = w_i$ for every $v_i \in \Sigma_A$,
- For $v_i, v_j \in X$, it holds that $w_i = w_j$ iff $v_i = v_j$, and $w_i, w_j \notin \Sigma_A$, and
- For $v_i = y$ and $v_j \neq y$, it holds that $w_i \neq w_j$.

Intuitively, a legal instance of v leaves all occurrences of $v_i \in \Sigma_A$ unchanged, associates all occurrences of $v_j \in X$ with the same unique letter, not in Σ_A, and associates every occurrence of y freely with letters from $\Sigma \setminus \Sigma_A$, different from those associated with X variables.

We say that a word $v \in \Gamma_A^*$ is a *witnessing pattern* for a word $w \in \Sigma^*$ if w is a legal instance of v. Note that v may be the witnessing pattern for infinitely many words in Σ^*, and that a word in Σ^* may have several witnessing patterns (or have none). Given a word $w \in \Sigma^*$, a *run of* $\mathcal{A}$ *on* w is a run of A on a witnessing pattern for w. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words in Σ^* for which there exists a witnessing pattern in $L(A)$.

Example 1. Let $\mathcal{A}_2 = \langle \Sigma, A_2 \rangle$ where A_2 is the automaton appearing in Figure 1. Then, $L(\mathcal{A}_2)$ is the language of all words in Σ^* in which some letter appears at least twice. By deleting the x_1 labels from the self loops in A_2, we get the language of all words in which some letter appears exactly twice.

Example 2. Let $\mathcal{A}_3 = \langle \Sigma, A_3 \rangle$ where A_3 is the NFA appearing in Figure 1. Then $L(\mathcal{A}_3)$ is the language of all words in Σ^* in which the last letter is different from all the other letters.

2.1. Comparison with Other Models

A classical model for recognizing languages over infinite alphabets is *finite state machines* (FMA) [11]. In this model, a finite set of registers are assigned letters from the input word during the run. The contents of the registers are distinct, and the transition function can refer to them. In [10], the authors

introduce NFMA, a nondeterministic extension of FMA in which the content of the registers may be updated nondeterministically during the run and needs not be restricted to letters in the input word.

VFAs are strictly less expressive than NFMA and are incomparable with FMA. Intuitively, the variables of a VFA are analogous to registers, but while a register can change its content during the run, a bounded variable cannot change the value assigned to it. Formally, a VFA $\mathcal{A} = \langle \Sigma, A \rangle$ with c constant letters and width d can be simulated by an NFMA with $c + d + 1$ registers. The first c registers are preassigned the constant letters and do not change their content during the run. The next d registers are preassigned nondeterministically and do not change their content during the run. Finally, the last register, which simulates the free variable, may change its content freely during the run.

To see the advantage that the ability to change the content of registers gives FMA, consider the language $L = \{\sigma_1\sigma_1\sigma_2\sigma_2 \dots \sigma_n\sigma_n : n \geq 0, \sigma_i \in \Sigma\}$. It is not hard to prove that the number of variables that a VFA needs in order to recognize L depends on the length of the input word, and is therefore unbounded. On the other hand, an NFMA (in fact, an FMA) for L can use a single register that changes its content all even positions. The ability to assign to the registers letters that have not been read yet is crucial for the simulation of VFA by NFMA. For example, as shown in [11], no FMA exists for the language $L(\mathcal{A}_3)$ from Example 2.

Data automata [2] run on *data words*. These are words over $\Sigma \times F$, where Σ is an infinite alphabet and F is n finite alphabet. A data automaton comprises two components. The first is a letter to letter transducer $\mathcal{B}$ which runs on the F projection of an input word w and outputs a word w' over an alphabet Γ. The second is an NFA $\mathcal{C}$, which runs on the subwords of w' corresponding to positions in w in which all letters of Σ are equal. Thus, the letters from Σ are not accessed directly, and are used for inducing an equivalence relation on the set of positions in w. The word w is accepted iff $\mathcal{B}$ accepts the F projection of w and $\mathcal{C}$ accepts all appropriate subwords of w'.

Data automata can accept the language $\{w | w_i \neq w_j \forall (i, j < |w|)\}$, which VFA cannot. Data automata cannot handle constants in Σ, as only equality checks are performed on the Σ projection of w. Therefore, data automata are incomparible with VFA. However, every constant-free VFA $\mathcal{A}$ over an infinite alphabet Σ with variables $X \cup \{y\}$ can be simulated by a data automaton $\mathcal{A}'$ over $\Sigma \times X \cup \{y\}$ that accepts a word w iff the $X \cup \{y\}$ projection of w is a witnessing pattern for the Σ projection of w in $\mathcal{A}$ as follows. The transducer is almost similar to the pattern automaton of $\mathcal{A}$. It outputs the $X \cup \{y\}$ projection of the input word w, with a minor change (resulting in a linear blow-up) of substituting the first appearance of x_i with x_i'. This substitution makes sure that every appearance of x_i is paired with the same letter the first appearance of x_i is paired with. The class automaton then consists of NFAs for $x'x^*$ for every $x \in X$, and an NFA for y^*.

3. Closure properties for VFAs

In this section we study closure properties of VFAs and the decidability and complexity of basic problems about them. We show that while VFAs are closed under union and intersection, they are not closed under complementation. In the computability front, we show that while the emptiness problem for VFAs is not harder than the one for NFA, the membership problem is harder, but still decidable, whereas the universality and containment problems are undecidable.

Theorem 1. *VFAs are closed under union.*

Proof: Let $\mathcal{A}_1 = \langle \Sigma, A_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, A_2 \rangle$ be VFAs with $A_1 = \langle \Sigma_1 \cup X_1 \cup \{y_1\}, Q_1, Q_0^1, \delta_1, F_1 \rangle$ and $A_2 = \langle \Sigma_2 \cup X_2 \cup \{y_2\}, Q_2, Q_0^2, \delta_2, F_2 \rangle$. [3]

A union construction for VFAs cannot simply guess whether to follow $\mathcal{A}_1$ or $\mathcal{A}_2$. The reason is that the set of constant letters of the union contains both Σ_1 and Σ_2. Therfore, while the variables of $\mathcal{A}_1$ and $\mathcal{A}_2$ range over $\Sigma \setminus \Sigma_1$ and $\Sigma \setminus \Sigma_2$, respectively, their occurrences in a simple union construction would range over $\Sigma \setminus (\Sigma_1 \cup \Sigma_2)$. Thus, a simple union construction may miss words in $L(\mathcal{A}_1)$ in which variables in X_1 are assigned values in Σ_2 and dually for $L(\mathcal{A}_2)$. The same problem exists with the free variables y_1 and y_2, whose range may be restricted in a simple union construction.

We solve this problem by defining the union of $\mathcal{A}_1$ and $\mathcal{A}_2$ as a union of several copies of the simple union. In every copy, a subset of bounded variables of one VFA is assigned a subset of constant letters of the other VFA.

Formally, let H_1 be the set of all one-to-one functions from subsets of X_1 to $\Sigma_2 \setminus \Sigma_1$. Each function in H_1 can be viewed as $h : X_1 \to (\Sigma_2 \setminus \Sigma_1) \cup \{\top\}$ such that for all $x, x' \in X_1$, if $h(x) = h(x')$ then $h(x) = \top$. Intuitively, $h^{-1}(\top)$ is the set of bounded variables that are not assigned values in $\Sigma_2 \setminus \Sigma_1$, and which are therefore not problematic. The function h then induces a bijection from the set of bounded variables that are not assigned $\top$ by h to $img(h) \setminus \{\top\}$.

Let A_1^h be the NFA obtained from A_1 by replacing all occurrences of x such that $h(x) \neq \top$ by $h(x)$, and replacing all occurrences of y_1 by $y \cup (\Sigma_2 \setminus img(h))$. That is, we add to every y_1 transition the set of constant letters that have not been assigned to the variables in h. Let A_1' be the union of the NFAs A_h^1, for all $h \in H_1$. Note that the alphabet of A_1' is $\Sigma_1 \cup X_1 \cup \{y\}$. We define H_2 and A_2' over $\Sigma_2 \cup X_2 \cup \{y\}$ in a similar way. Now, $\mathcal{A} = \langle \Sigma, A \rangle$, where A is the union of A_1' and A_2'.

If the number of states, width, and number of constant letters in $\mathcal{A}_1$ and $\mathcal{A}_2$ are n_1, n_2, d_1, d_2, c_1 and c_2, respectively, then we can bound the number of states of $\mathcal{A}$ by $n_1 c_2^{d_1+1} + n_2 c_1^{d_2+1}$, and its width by $d_1 + d_2$. These bounds follow from an analysis of the number of different functions H_1 and H_2. The analysis can be made tighter. In particular, note that if $\Sigma_1 = \Sigma_2$, the simple union construction works. □

[3] For simplicity, both $\mathcal{A}_1$ and $\mathcal{A}_2$ are over the same alphabet Σ, but it is possible, with minor modifications, to construct the union (and intersection) for the case where the two VFAs are over different alphabets.

Theorem 2. *VFA are closed under intersection.*

Proof: Let $\mathcal{A}_1 = \langle \Sigma, A_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, A_2 \rangle$ be VFAs with $A_1 = \langle \Sigma_1 \cup X_1 \cup \{y_1\}, Q_1, Q_0^1, \delta_1, F_1 \rangle$ and $A_2 = \langle \Sigma_2 \cup X_2 \cup \{y_2\}, Q_2, Q_0^2, \delta_2, F_2 \rangle$.

Recall that in the product construction for the NFAs A_1 and A_2, the state space is $Q_1 \times Q_2$, and $\langle q_1', q_2' \rangle \in \delta(\langle q_1, q_2 \rangle, a)$ iff $q_1' \in \delta_1(q_1, a)$ and $q_2' \in \delta_2(q_2, a)$. Since A_1 and A_2 are pattern automata of VFAs, the letter a may be a variable. Accordingly, there are cases in which it should be possible to intersect two differently labeled transitions: intersecting two transitions with differently named bounded variables, meaning they get the same assignment in $\mathcal{A}_1$ and in $\mathcal{A}_2$; intersecting a variable with a letter σ, meaning the variable is assigned σ; and intersecting the free variable y with a bounded variable x or with a letter σ, meaning the assignment to y in this transition agrees with the assignment of x or with σ.

Accordingly, we would like to define δ such that for $z \in (\Sigma_1 \cup \Sigma_2) \cup X \cup \{y\}$, we have that $\langle q_1', q_2' \rangle \in \delta(\langle q_1, q_2 \rangle, z)$ iff there exist $z_1 \in \Sigma_1 \cup X \cup \{y\}$ and $z_2 \in \Sigma_2 \cup X \cup \{y\}$ such that $q_1' \in \delta_1(q_1, z_1)$ and $q_2' \in \delta_2(q_2, z_2)$ and such that z_1 and z_2 can be matched according to the cases described above.

In order to define δ, we define a relation H for matching the variables and constant letters of $\mathcal{A}_1$ with the variables and constant letters of $\mathcal{A}_2$. As there may be several possible such relations, the intersection automaton is a union of several VFAs, one for every relation.

Formally, consider a relation $H_V \subseteq (X_1 \cup \{y_1\}) \times (X_2 \cup \{y_2\})$ and a relation $H_{AB} \subseteq ((\Sigma_1 \setminus \Sigma_2) \times (X_2 \cup \{y_2\})) \cup (X_1 \cup \{y_1\}) \times (\Sigma_2 \setminus \Sigma_1)$ such that every $x \in X_1 \cup X_2$ and every $\sigma \in (\Sigma_1 \cup \Sigma_2)$ appears in at most one pair in $H_V \cup H_{AB}$. Thus, the relation H_{AB} is for the cases in which in one VFA the transition is labeled by a constant letter a, and in the other the transition is labeled by a variable z, and z is assigned a. The relation H_V is for the cases in which in both VFAs the transitions are labeled by variables, and these variables are equally assigned. Hence, H_V defines the variables of the intersection construction: a pair $(z_1, z_2) \in H_V$ in which z_1 or z_2 is bounded is a bounded variable, and (y_1, y_2) is the free variable. Let $H = H_V \cup H_{AB}$.

We define $A_H = \langle (\Sigma_1 \cup \Sigma_2) \cup H_V, Q_1 \times Q_2, Q_0^1 \times Q_0^2, \delta, F_1 \times F_2 \rangle$, where $\langle q_1', q_2' \rangle \in \delta(\langle q_1, q_2 \rangle, z)$ if one the following holds.

- $z \in (\Sigma_1 \cap \Sigma_2), q_1' \in \delta_1(q_1, z)$, and $q_2' \in \delta_2(q_2, z)$,
- $z \in \Sigma_2$ and there is z_1 such that $\langle z_1, z \rangle \in H_{AB}$, $q_1' \in \delta_1(q_1, z_1)$, and $q_2' \in \delta_2(q_2, z)$,
- $z \in \Sigma_1$ and there is z_2 such that $\langle z, z_2 \rangle \in H_{AB}, q_1' \in \delta_1(q_1, z)$, and $q_2' \in \delta_2(q_2, z_2)$,
- $z = \langle z_1, z_2 \rangle, \langle z_1, z_2 \rangle \in H_V, q_1' \in \delta_1(q_1, z_1)$, and $q_2' \in \delta_2(q_2, z_2)$.

We define $\mathcal{A} = \langle \Sigma, \bigcup_H A_H \rangle$. The set of bounded variables of $\mathcal{A}$ is $H_V \setminus \{\langle y_1, y_2 \rangle\}$, and the free variable is $\langle y_1, y_2 \rangle$.

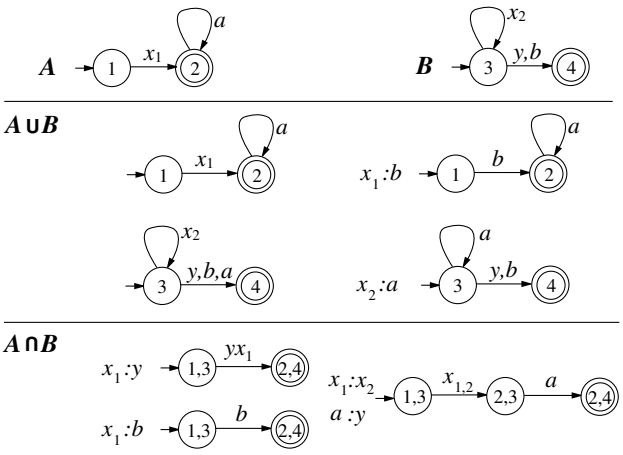

Figure 2: The VFAs A and B, and their union and intersection constructions.

If the number of states and width, and number of constant letters in $\mathcal{A}_1$ and $\mathcal{A}_2$ are n_1, n_2, d_1, d_2, c_1 and c_2, respectively, then we can bound the number of states of $\mathcal{A}$ by $O((n_1 n_2)\frac{(d_1+d_2+c_1+c_2)!}{(c_1+c_2)!})$, and its width by $O(\frac{(d_1+d_2+c_1+c_2)!}{(c_1+c_2)!})$.

The bound on the size of $\mathcal{A}$ follows from an analysis of the number of different relations H.

□

Example 3. Figure 2 shows the union and intersection constructions for the VFAs A and B.

Theorems 1 and 2 show that while the straightforward union and product constructions for NFAs do not work for VFAs, VFAs are still closed under union and intersection via more complicated constructions. We now show that the same cannot be done for the problem of complementation.

Theorem 3. *VFAs are not closed under complementation.*

Proof: Consider the VFA $\mathcal{A}_2$ of Example 1. Recall that $L(\mathcal{A}_2)$ contains exactly all words in Σ^* in which some letter appears at least twice. The complement $\tilde{L}$ of $L(\mathcal{A}_2)$ then contains exactly all words all of whose letters are pairwise distinct.

Assume by way of contradiction that there exists a VFA $\bar{\mathcal{A}}_2$ of width k that recognizes $\tilde{L}$. Let w be a word of length $k+2$ whose letters are pairwise distinct and does not contain a constant letter of $\bar{\mathcal{A}}_2$. Then $w \in \tilde{L}$. Let v be a witnessing pattern for w in $\tilde{L}$. Then v contains no constant letters. Also, it cannot contain two bounded variables, as they are equally assigned, and hence must contain at least two occurences of the free variable y. Therefore, a legal assignment for v

in $\tilde{L}$ that assigns all occurences of y the same letter a creates a word $g(v) \in \tilde{L}$ in which a appears more than once, a contradiction.

□

4. Decision procedures for VFAs

The basic decision problems for automata are membership (given $\mathcal{A}$ and a word $w \in \Sigma^*$, is $w \in L(\mathcal{A})$?), nonemptiness (given $\mathcal{A}$, is $L(\mathcal{A}) \neq \emptyset$?), universality (given $\mathcal{A}$, is $L(\mathcal{A}) = \Sigma^*$?) and containment (given $\mathcal{A}_1$ and $\mathcal{A}_2$, is $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$?). We now turn to study the decidability and complexity of these problems for VFAs.

Theorem 4. *The nonemptiness problem for VFA is NL-complete.*

Proof: Let $\mathcal{A} = \langle \Sigma, A \rangle$. We prove that $L(\mathcal{A})$ is nonempty iff $L(A)$ is nonempty. Since NFA nonemptiness is NL-complete, both the upper and lower bounds would follow. For the first direction, consider $w \in L(\mathcal{A})$. By the way we defined $L(\mathcal{A})$, there exists a word v in $L(A)$ and a legal assignment g for v such that $g(v) = w$. Therefore, $L(A)$ is nonempty. For the other direction, consider $v \in L(A)$. Since Σ is infinite, there exists a legal assignment g for v, inducing a word in $L(\mathcal{A})$. □

Theorem 5. *The membership problem for VFA is NP-complete.*

Proof: For the upper bound, consider a VFA $\mathcal{A} = \langle \Sigma, A \rangle$ and a word $w \in \Sigma^*$. A nondeterminitic polynomial-time algorithm for deciding whether $w \in L(\mathcal{A})$ guesses a word $v \in \Gamma_A^{|w|}$ and a legal assignment g for v in $\mathcal{A}$ over the letters of w, and checks whether $v \in L(A)$ and $g(v) = w$.

The lower bound is shown by a reduction from the Hamiltonian cycle problem for directed graphs. Let $G = \langle V, E \rangle$ be a directed graph with n vertices. We construct VFA $\mathcal{A}_G = \langle \mathbb{N}, A_G \rangle$ as follows. The pattern automaton $A_G = \langle V, V, \{v_1\}, \delta_G, \{v_1\} \rangle$ is such that the set of bounded variables is V, and if $E(v_i, v_j)$, then $\delta_G(v_i, v_j) = \{v_j\}$; otherwise $\delta_G(v_i, v_j) = \emptyset$. Thus, A_G is obtained from G by labeling each edge by its destination.

Consider the word $w = 1 \cdot 2 \cdot 3 \cdots n$. Since every variable must be assigned a different letter, a run on w matches a path in A_G of length n in which n different vertices have been traversed, the last of which is the initial vertex. This exactly matches a description of a Hamiltonian cycle in G.

□

So nonemptiness is not harder than the case of finite alphabets, and while membership is harder, it is still decidable. The picture is less nice when we turn to study the universality and containment problems. Here, the algorithms for the finite-alphabet case relies on the closure of NFAs under complementation, which does not hold for VFAs. As we show below, the problems are indeed undecidable.

Theorem 6. *The universality and containment problems for VFAs are undecidable.*

Proof: We start with the universality problem. In [13], the authors prove the undecidability of universality problem for register automata by a reduction from PCP (Post's Correspondence Problem).

The reduction translates a PCP instance, which is a pair of sets of words A and B, into an automaton that accepts an input word iff it is not a legal encoding of the PCP instance, or a legal encoding that does not represent a solution. More specifically, the solution is the encoding of the concatination of words of A, followed by a seperating mark and the encoding of the concatenation of the words of B. The two concatenations in a correct legal encoding are identical. The language of the automaton is then universal iff there exists no solution for the PCP instance.

The idea is to use a double indexing system based on unique data values. Every encoded word is preceded by a unique data value of one index, and every letter in every word is preceded by a unique data value of the other. This allowes the automaton to check that the two parts indeed form a correct solution, by checking that the two concatenations are indeed identical, and are legally formed according to the PCP rules. Such a system can be used also in VFAs, and the reduction is very similar. Since the PCP problem is undecidable, undecidability for universality follows. Since we can easily define a universal VFA, undecidability for the containment problem follows too. □

5. Deterministic VFA

In this section we define deterministic VFA and study their properties. We show that deterministic VFA are simple, expressive, and are closed under all Boolean operations. In addition, the nonemptiness, membership, universality, and containment problems are all decidable for them.

Recall that an NFA is deterministic if $|Q_0| = 1$ and for all $q \in Q$ and $\sigma \in \Sigma$, we have $|\delta(q, \sigma)| \leq 1$. Indeed, these syntactic conditions guarantee that the automaton has at most one run on each input word. To see that such a syntactic characterization does not exist for VFA, consider the VFA $\mathcal{A}$ appearing in Figure 3. Its pattern automaton is deterministic, but the word a has two different runs in $\mathcal{A}$: one in which x_1 is assigned a, and one in which x_2 is assigned a. Thus, there is a need to define deterministic VFAs in a non-syntactic manner.

Definition 1. A VFA $\mathcal{A} = \langle \Sigma, A \rangle$ is *deterministic* (DVFA, for short), if for every word $w \in \Sigma^*$, there exists exactly one run of $\mathcal{A}$ on w.

Note that, equivalently, a VFA is deterministic if for every word in Σ^* there is exactly one witnessing pattern, on which there is a single run in the pattern automaton.

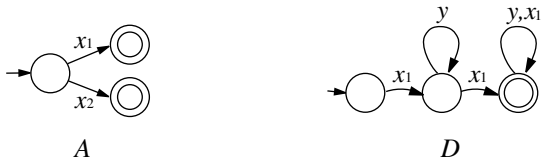

Figure 3: A nondeterministic VFA whose pattern automaton is deterministic, and an DVFA that accepts all words in which the first letter is repeated at least twice.

Example 4. Consider the VFA $\mathcal{D} = \langle \Sigma, D \rangle$, where D is the DFA appearing in Figure 3. The language of $\mathcal{D}$ is the set of all words over Σ in which the first letter is repeated at least twice. To see that it is deterministic, consider a word $w = w_1 w_2 \dots w_n$ in Σ^*. A witnessing pattern for w is over x_1 and y. Since only x_1 exits the initial state, then x_1 must be assigned w_1, and all other occurences of other letters must be assigned to y. Therefore, every word that has a witnessing pattern has a single witnessing pattern. Since D is deterministic, every witnessing pattern has a single run in D. It follows that $\mathcal{D}$ is deterministic.

Example 4 demonstrates the expressive power of deterministic VFAs. In fact, as we will show in Theorem 16, every VFA without a free variable has an equivalent DVFA.

Recall that deciding whether an NFA is deterministic is easy, as the definition of determinization is syntactic. For VFA, the definition is semantic. As we show below, however, an equivalent syntactic definition does exist.

Theorem 7. *Deciding whether VFA is deterministic is NL-complete.*

Proof: We start with the upper bound. Consider a VFA $\mathcal{A} = \langle \Sigma, A \rangle$ with variables $X \cup \{y\}$ and an initial state q_{in}. We claim that $\mathcal{A}$ is not deterministic iff one of the following holds.

1. A is nondeterministic,
2. there exists a reachable state s such that there exist two bounded variables x and x' that exit s, and a path from q_{in} that reaches s and does not traverse x and x',
3. there exists a bounded variable x such that both x and y exit s, and a path from q_{in} that reaches s but does not traverse x,
4. there exists a reachable state s such that there exists a constant letter that does not exit s, or a variable that appears along a path from q_{in} to s that does not exit s, or
5. there exists a path π from q_{in} to s such that all variables that exit s appear along π, and y does not exit s.

Intuitively, conditions 2 and 3 check that each word $w \in \Sigma^*$ has at most one run in $\mathcal{A}$. Indeed, if conditions 2 or 3 hold, then after reaching s, the next input letter may be assigned to both x and x' (in condition 2), or to x and y (in condition 3). If A is deterministic, these are the only conditions in which continuing along two transitions with the same input letter is possible.

Conditions 4 and 5 check that w has at least one run on every input word. Indeed, if condition 4 holds, then the run is stuck from s if the next input letter is the missing constant, or the letter that has been assigned to the missing variable. If condition 5 holds, then the run is stuck from s if the next input letter has not appeared before.

In order to implement the above check in NL, we guess the condition that is violated, and check that it is indeed violated. For example, to check the second item, we can guess the state s, a path π to s, and x and x', and check that π does not traverse x and x', and that both x and x' exit s. Since NL is closed under complementation, we are done.

For the lower bound, we do a reduction from the reachability problem for directed graphs. Given a directed graph G and two vertices s and t, consider the VFA $\mathcal{A}_G = \langle \mathbb{N}, A_G \rangle$, where A_G is obtained from G by labeling its edges by different constant letters and adding two transitions, both labeled x_1, from the state t to two new states. In addition, we add to A_G transitions to a rejecting sink from every state, labeled both by y and all constant letters that do not exit the state. It is not hard to see that the vertex t is reachable from the vertex s in G iff $\mathcal{A}_G$ is not detereministic.

□

Note that Theorem 7 refers to the problem of deciding whether a given VFA is deterministic and not whether it has an equivalent DVFA. As we show in the sequel, the latter problem is much harder.

The closure of DVFA under the operations of union and intersection does not follow from Theorems 1 and 2, as even if applied on DVFA, these constructions do not yield a DVFA.

In order to present the various constructions for DFVAs, we introduce an *unwinding operator* for VFA. Given a VFA over Σ with a pattern automaton $A = \langle \Sigma_A \cup X \cup \{y\}, Q, Q_0, \delta, F \rangle$, the *unwinding* of $\mathcal{A}$ is the VFA $\mathcal{U} = \langle \Sigma, U \rangle$, with $U = \langle \Sigma_A \cup X \cup \{y\}, Q \times 2^X, \langle Q_0, \emptyset \rangle, \rho, F \times 2^X \rangle$, where ρ is defined, for every $\langle q, \theta \rangle \in Q \times 2^X$ and $z \in \Sigma_A \cup X \cup \{y\}$ as follows.

$$\rho(\langle q, \theta \rangle, z) = \begin{cases} \delta(q, z) \times \{\theta \cup \{z\}\} & z \in X \cup \{y\} \\ \delta(q, z) \times \{\theta\} & z \in \Sigma_A \end{cases} \tag{1}$$

Intuitively, the states in $\mathcal{U}$ keep track of the set of variables that have been traversed (and hence also assigned, in case of bounded variables) along the path from the initial state.

Remark 1. A state in the unwinding reflects the set of all variables that are read until reaching this state. The conditions in the proof of Theorem 7 refer to

this set. Consequently, for determining whether an unwound VFA is deterministic, checking these sets is enough. An unwound automaton $\mathcal{U}$ with a pattern automaton U is a DVFA iff all the following conditions hold:

1. U is deterministic
2. From every state $\langle q, \theta \rangle$ exits either a single bounded variable not in θ, or y, but not both.
3. From every state $\langle q, \theta \rangle$ exit all the variables in θ and all constant letters.

These conditions follow from the conditions in the proof of Theorem 7.

Lemma 1. *A VFA is equivalent to its uwinding.*

Proof: Consider a VFA $\mathcal{A} = \langle \Sigma, A \rangle$ and its unwinding $\mathcal{U} = \langle \Sigma, U \rangle$. Let $w \in L(\mathcal{A})$, and let $q_0, q_1, \ldots, q_m$ be an accepting run of the pattern automaton A of $\mathcal{A}$ on a witnessing pattern $v = v_1 v_2 \ldots v_m$ for w in $\mathcal{A}$. Then by the way we have defined ρ, the run $\langle q_0, \emptyset \rangle, \langle q_1, \theta_1 \rangle, \ldots, \langle q_m, \theta_m \rangle$ where θ_i is the set of all variables that appear in the prefix $v_1 v_2 \ldots v_i$ of v is an accepting run of $\mathcal{U}$ on w.

Now, let $w \in L(\mathcal{U})$, and let $\langle q_0, \emptyset \rangle, \langle q_1, \theta_1 \rangle, \ldots, \langle q_m, \theta_m \rangle$ be an accepting run of $\mathcal{U}$ on a witnessing pattern v of w. Then, by the way we have defined ρ, the run $q_0, q_1, \ldots, q_m$ is an accepting run of A on v, and therefore $w \in L(\mathcal{A})$. □

Lemma 2. *A VFA is deterministic iff its unwinding is deterministic.*

Proof: Consider a VFA $\mathcal{A} = \langle \Sigma, A \rangle$ and its undinding $\mathcal{U} = \langle \Sigma, U \rangle$. We first prove that if $\mathcal{A}$ is deterministic then $\mathcal{U}$ is deterministic. Assume by way of contradiction that $\mathcal{A}$ is deterministic and $\mathcal{U}$ is not. Then one of the conditions of the proof of Theorem 7 applies. We now go over the different conditions and show that they all lead to a contradiction.

Condition 1. According to the way we defined $\mathcal{U}$, we have that if A is deterministic then U is deterministic.

Conditions 2-4. Let $\pi = \langle q_0, \emptyset \rangle, \langle q_1, \theta_1 \rangle \ldots \langle q_m, \theta_m \rangle$ be the path in $\mathcal{U}$ that is mentioned in conditions 2-4, reading a word v. According to the way we defined $\mathcal{U}$, we have that $q_0, q_1, \ldots, q_m$ is a path in A reading v, and that the set of labels exiting $\langle q_m, \theta_m \rangle$ is the set of labels exiting q_m. Therefore, the state q_m in A fulfills conditions 2-4, and so $\mathcal{A}$ is nondeterministic, a contradiction.

Similarly, to show that if $\mathcal{U}$ is deterministic then $\mathcal{A}$ is deterministic, we go over the conditions in the proof of Theorem 7 and show that assuming otherwise leads to a contradiction.

Condition 1. According to the way we defined $\mathcal{U}$, we have that if U is deterministic then A is deterministic.

Conditions 2-4. Let $\pi = q_0, q_1, \ldots, q_m$ be the path in A that is mentioned in conditions 2-4, and let $v = v_1 v_2 \cdots v_m$ be the word read along π. Then π matches a path $\langle q_0, \emptyset\rangle, \langle q_1, \theta_1\rangle, \ldots, \langle q_m, \theta_m\rangle$ reading v in $\mathcal{U}$, where θ_i is the set of variables appearing in $v_1 v_2 \ldots v_i$. The set of labels that exit q_m is the set of labels that exit $\langle q_m, \theta_m\rangle$. Therefore, the state $\langle q_m, \theta_m\rangle$ in U fulfills conditions 2-4, and so $\mathcal{U}$ is nondeterministic, a contradiction.

□

We now present the constructions for union and intersection. The constructions have the construction for DFAs in their basis, applied to the unwinding of the DVFA. As in Theorems 1 and 2, there is a need to match the variables and constant letters of one DVFA with those of the other. The properties of the DVFA and the unwinding induce a deterministic matching.

Theorem 8. *DVFA are closed under intersection.*

Proof: Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be DVFAs over an alphabet Σ [4]. Let $\mathcal{U}_1$ and $\mathcal{U}_2$ be the unwindings of $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. Recall from Remark 1 that from every state $s = \langle q, \theta\rangle$ in $\mathcal{U}_1$ (resp. $\mathcal{U}_2$) there exits either a bounded variable not in θ or y, and all the variables in θ and constant letters exit s. We construct a VFA $\mathcal{U}$ over Σ such that $L(\mathcal{U}) = L(\mathcal{U}_1) \cap L(\mathcal{U}_2)$ in which this properties also hold, and is identical to its unwinding. Therefore, the VFA $\mathcal{U}$ is deterministic.

Recall that the construction for intersecting two VFAs involves matching the variables and constants of one VFA with the variables and constant letters of the other. In the case of VFA, several such matchings are possible. In the case of DVFA, a similar matching is constructed, induced from the properties of Remark 1. Since DVFA are full, there is a run on every word and so it is always possible to continue along both automata simultaneously. However, the result must remain deterministic.

Intuitively, the DVFA $\mathcal{U}$ is constructed by matching the new variables that exit a pair of states in $\mathcal{U}_1 \times \mathcal{U}_2$ with each other or with the free variable, and with constant letters. These matchings are remembered by the states that follow.

More specifically, consider a state q in the construction that represents a pair of states $\langle s, t\rangle \in \mathcal{U}_1 \times \mathcal{U}_2$. If s and t introduce new variables x_1 and x_2, respectively, then these variables are matched together to form a new variable $\langle x_1, x_2\rangle$ that exits q. If only s (w.l.o.g.) introduces a new variable x_1, then according to Remark 1 we have that y_2 exits t, and x_1 and y_2 are matched together to form a new variable $\langle x_1, y_2\rangle$ that exits q. Finally, if y_1 exits s and y_2 exits t then they are matched together and form the free variable y. Additionally, new variables are matched with every constant letter that is not a constant in their own automaton. All these matchings are remembered by

[4] For simplicity, both $\mathcal{D}_1$ and $\mathcal{D}_2$ are over the same alphabet Σ, but it is possible, with minor modifications, to construct the union and intersection for the case where the two DVFAs are over different alphabets.

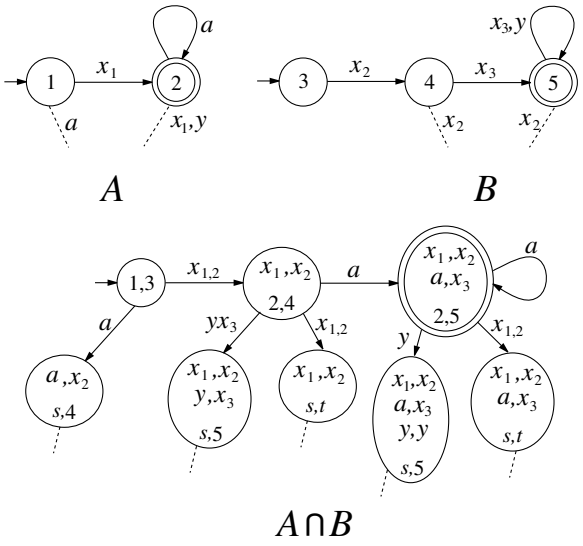

Figure 4: The DVFAs A and B, and their intersection $A \cap B$. s and t denote rejecting sinks in A and B. The dotted lines lead to further rejecting states and sinks.

the states that are reachable from q by augmenting every state with the set of matchings that have been made until it is reached.

The complete transition relation and proof of correctness is given in Appendix A.1

□

Theorem 9. *DVFA are closed under union.*

Proof: As in intersection of DVFA, the construction for the union is performed on top of the unwindings. Since DVFA are full, there is a run on every word and so it is always possible to continue along both automata simultaneously. Consequently, the construction for the union is similar to that of the intersection. The only difference is that here, a state representing a pair $\langle s, t \rangle$ is accepting if either s or t are accepting. The proof of correctness is in Appendix A.2.

□

Example 5. Figure 4 shows the result of the intersection of two DVFAs with pattern automata A and B, where a is the only constant letter of A and B has no constant letters. For convenience, only some of the rejecting states and sinks are shown. The union construction has a similar construction, with more states marked as accepting.

The fact that a DVFA has exactly one run on each input word makes its complementation easy: one only has to complement the pattern automaton. Formally, we have the following.

Theorem 10. *DVFAs are closed under complementation. Given a DVFA $\mathcal{A}$, we can construct a DVFA $\tilde{A}$ with the same size and width such that $L(\tilde{\mathcal{A}}) = \Sigma^* \setminus L(\mathcal{A})$.*

Proof: Let $\mathcal{A} = \langle \Sigma, A \rangle$. Consider the DFA $\tilde{A}$ that dualizes A. That is, if $A = \langle \Gamma_A, Q, q_0, \delta, F \rangle$, then $\tilde{A} = \langle \Gamma_A, Q, q_0, \delta, Q \setminus F \rangle$. Let $\tilde{\mathcal{A}} = \langle \Sigma, \tilde{A} \rangle$. We claim that $L(\tilde{\mathcal{A}}) = \Sigma^* \setminus L(\mathcal{A})$.

To see this, consider a word $w \in \Sigma^*$. Let $v \in \Gamma_A$ be the unique witnessing pattern of w in A. The word w is in $L(\mathcal{A})$ iff $v \in L(A)$. Since $L(\tilde{A}) = \Gamma_A^* \setminus L(A)$, the latter holds iff $v \notin L(\tilde{A})$, which holds iff $w \notin L(\tilde{\mathcal{A}})$. □

We now turn to study the complexity of the DVFA model. We first study the problems of nonemptiness and membership. As argued in the proof of Theorem 4, a VFA is empty iff its pattern automaton is empty. Since the nonemptiness problem is NL-complete also for DFAs, the NL-complete complexity there applies also for DVFAs. For the membership problem we have to describe a more complicated lower bound.

Theorem 11. *The membership problem for DVFA is in PTIME.*

Proof: Consider a DVFA $\mathcal{D} = \langle \Sigma, D \rangle$ and a word $w = w_1 w_2 \cdots w_n$. We can check the membership of w in $L(\mathcal{D})$ by simulating a run of $\mathcal{D}$ on it. In the process, we maintain a list of all the assignments that have been made so far, and consult the list in order to resolve branches labeled by different variables in D. □

We note that the question of whether the membership problem is PTIME-hard, or in NL is still open, and we suspect that it is very difficult, as it has the same flavor of the long-standing open problem of the complexity of one-path LTL model checking [12]. In both problems, one has to go back and forth a single path, and it is not clear whether it is possible to bound by a logarithmic function the information he has to store during this traversal.

We now turn to study the universality and containment problems and show that they are decidable.

Theorem 12. *The universality problem for DVFA is NL-complete.*

This result follows from the NL-completeness of the emptiness problem, and from the fact that complementation only involves a dualization of the acceptance condition.

Since DVFA are closed under complementation and instersection, the containment problem is also decidable. In fact, we have the following.

Theorem 13. *The containment problem for DVFA is in co-NP.*

Proof: Consider two DFVAs $\mathcal{A}_1$ and $\mathcal{A}_2$. We claim that if there is a word in $L(\mathcal{A}_1) \setminus L(\mathcal{A}_2)$ (that is, a witness that $\mathcal{A}_1$ is not contained in $\mathcal{A}_2$), then there is

also such a word whose length is bounded by $n_1 \cdot n_2$, where n_1 and n_2 are the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. By Theorem 11, checking such a witness can be done in PTIME.

To prove the claim about the length of the witness, let us consider the VFA for $L(\mathcal{A}_1) \setminus L(\mathcal{A}_2)$. We can construct such a VFA by complementing $\mathcal{A}_2$ (by dualizing the acceptance condition of its pattern automaton) and then taking its product, as specified in the proof of Theorem 2, with $\mathcal{A}_1$. Note that we define the product as a VFA, rather than a DVFA, so no unwinding is required. While the product may contain several copies of the basic product constructions, its diameter is bounded by $n_1 \cdot n_2$, and hence it is not empty iff there is a witness of length at most $n_1 \cdot n_2$ to its nonemptiness. □

6. Determinization

In this section we show that not all VFAs have an equivalent DVFA, and the problem of determinizing a given VFA (or concluding that no equivalent DVFA exists) is undecidable. As good news, we point to a fragment of VFAs that can always be determinized.

One evidence that not all VFAs have an equivalent DVFA is the fact that while DVFA are closed under complementation, VFA are not. As a specific example, which also demostrates the weakness of DVFA, consider the VFA $\mathcal{A}_2$ of Example 1. The language of $\mathcal{A}_2$ is the set of all words in which some letter is repeated at least twice. We claim that it has no equivalent DVFA. To see this, assume by way of contradiction that it has an equivalent DVFA $\mathcal{D}$ of width d. Let w be a word of length $d + 2$ whose letters are pairwise distinct and contains no constant letters. Then w is not accepted by $\mathcal{D}$. By the way we defined w, the witnessing pattern v of w must include at least two occurences of y. By assigning these two occurences of y the same letter, we get a word w' that should accepted by $\mathcal{D}$. However, its witnessing pattern v is rejected by the pattern automaton, a contradiction.

Theorem 14. *The problem of determinizing a given VFA (or concluding that no equivalent DVFA exists) is undecidable.*

Proof: Assume by way of contradiction that there is a Turing Machine M that, given a VFA, returns an equivalent DVFA or announces that no such DVFA exists. We construct from M a Turing machine M' that decides the universality problem for VFA, which, according to Theorem 6, is undecidable.

The machine M' proceeds as follows. Given a VFA $\mathcal{A}$, it runs M on $\mathcal{A}$. If M returns that $\mathcal{A}$ does not have an equivalent DVFA, then M' returns that $\mathcal{A}$ is not universal. This, since a single accepting state with a self loop labeled y is a universal DVFA and could be returned by M'. Otherwise, M' returns a DVFA $\mathcal{A}'$ equivalent to $\mathcal{A}$. By Theorem 12, M' can then check $\mathcal{A}'$ for universality. □

However, there exist fragments of VFA for which a determinization process does exist.

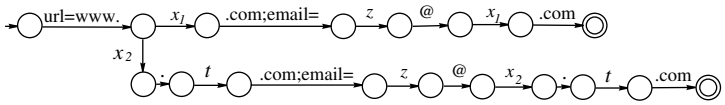

Figure 5: A syntactically determinizable VFA

Definition 2. We say that a VFA is *syntactically determinizable* if it contains no y transitions.

For example, consider the syntactically determinizable VFA $\mathcal{A} = \langle \{\mathrm{a}, \ldots, \mathrm{z}\}^*, A \rangle$, appearing in Figure 5. The VFA $\mathcal{A}$ accepts all words of the form

`url=www.`x_1`.com;email=`z`@`x_1`.com` or `url=www.`x_2`.t.com;email=`z`@`x_2`.t.com`,

where x_1, x_2, t, and z are words over the alphabet $\{\mathrm{a}, \ldots, \mathrm{z}\}$. Thus, $\mathcal{A}$ makes sure that the domain of the url agrees with that of the email, and it nondeterministically branches to allow both domain of the form x`.com` and of the form x`.t.com`.

Theorem 15. *A syntactically determinizable VFA has an equivalent DVFA.*

For every syntactically determinizable VFA there exists an equivalent DVFA, which is obtained by a determinization process we describe next.

6.1. A determinization procedure for syntactically determinizable VFA

Let $A = \langle \Sigma_A \cup X \cup \{y\}, q_0, Q, \delta, F \rangle$ be the pattern automaton of $\mathcal{A}$, and let $\mathcal{U}$ and $U = \langle \Sigma_A \cup X \cup \{y\}, \langle q_0, \emptyset \rangle, Q \times 2^X, \rho, F \times 2^X \rangle$, be the unwinding of $\mathcal{A}$ and the pattern automaton of the unwinding of $\mathcal{A}$, respectively. We assume that A is deterministic, and therefore U is deterministic. It is easy to see that in case that A is not deterministic, applying the subset construction on $\mathcal{A}$ yields an equivalent VFA (but not necessarily a DVFA).

It follows from the proof of Theorem 7 that in case that A is deterministic and y does not appear in the pattern automaton, a word may have more than one run in the unwinding $\mathcal{U}$ iff there exists a state $s = \langle q, \theta \rangle$ in $\mathcal{U}$ such that more than one variable not in θ exits s. Intuitively, the DVFA $\mathcal{D}$ is formed from $\mathcal{U}$ by grouping together all transitions labeled by new variables exiting the different states and substituting them with a single transition labeled by a fresh variable. Then, in a similar manner to the subset construction, all states these transitions reach are grouped together. Finally, rejecting sinks and transitions are added in order to have a single run on all words not in $L(\mathcal{A})$.

Formally, we construct an equivalent DVFA $\mathcal{D}$ with a pattern automaton D, a set of bounded variables Z, a set of states Q_D, a set of accepting states F_D and a transition function δ_D as follows. During the procedure we rename labels in X by fresh variables that we add to Z. A state in Q_D is of the form $\{\langle q_1, \eta_1 \rangle, \langle q_2, \eta_2 \rangle, \ldots, \langle q_m, \eta_m \rangle\}$ where $q_i \in Q$ and $\eta_i \subseteq (X \times Z)$. The state is accepting iff there exists $1 \leq i \leq m$ s.t. $q_i \in F$. Let $\eta_i|_X = \{x | \langle x, z \rangle \in \eta_i\}$, and similarly, let $\eta_i|_Z = \{z | \langle x, z \rangle \in \eta_i\}$. Then, for a state $\langle q_i, \eta_i|_X \rangle \in Q_U$, the set η_i

denotes the set of variables in $\eta_i|_X$ and their mapping to their new names in Z. It is possible that $q_i = q_j$ for $i \neq j$, if $\eta_i \neq \eta_j$.

We begin the procedure by adding the initial state $\{\langle q_0, \emptyset\rangle\}$ to Q_D. At every iteration, for every state $s = \{\langle q_1, \eta_1\rangle, \langle q_2, \eta_2\rangle, \ldots, \langle q_m, \eta_m\rangle\} \in Q_D$, let

$$X_s = \bigcup_{\langle q_i, \eta_i\rangle \in s} \{x | x \in X \text{ and there exists } \langle q, \theta\rangle \in \rho(\langle q_i, \theta_i\rangle, x), \text{ where } \theta_i = \eta_i|_X\}.$$

Then X_s is the set of variables exiting the states $\langle q_i, \eta_i|_X\rangle$ for $1 \leq i \leq m$ in $\mathcal{U}$. Let

$$X'_s = \bigcup_{\langle q_i, \eta_i\rangle \in s} \{x | x \in X \text{ and there exists } \langle q, \theta\rangle \in \rho(\langle q_i, \theta_i\rangle, x) \text{ such that } x \notin \theta_i, \text{ where } \theta_i = \eta_i|_X\}.$$

Then X'_s is the set of variables that exit some state in s which does not know them yet. We group the variables in X'_s together and rename them by a fresh variable z we add to Z. For this, we add new states to $\mathcal{D}$ as described next.

For a new $z \in Z$ induced by a state s we add a new state s' to $\mathcal{D}$ as follows. For every $\langle q, \eta\rangle \in s$ and $\langle q', \theta\rangle \in Q_U$ such that $\langle q', \theta\rangle \in \rho(\langle q, \eta|_X\rangle, x)$ for $x \in X'_s$, we add $\langle q', \eta \cup \{\langle x, z\rangle\}\rangle$ to s'. We define $\delta_D(s, z) = s'$. Notice that by the definition of ρ we have that $\theta = \eta|_X \cup \{x\}$, and so adding $\langle q', \eta \cup \{\langle x, z\rangle\}\rangle$ to s' reflects the transition from $\langle q, \eta|_X\rangle$ to $\langle q', \theta\rangle$, and adds the new mapping $\langle x, z\rangle$ to the list of mappings η.

For every other $z \in Z$, we add a new state s' to $\mathcal{D}$, if it does not already exist, as follows. For every $\langle q, \eta\rangle \in s$ such that $\langle x, z\rangle \in \eta$ for some $x \in X_s$ and $\langle q', \theta\rangle \in Q_U$ such that $\langle q', \theta\rangle \in \rho(\langle q, \eta|_X\rangle)$, we add $\langle q', \eta\rangle$ to s'. We define $\delta_D(s, z) = s'$. Notice that by the definition of ρ we have that $\theta = \eta|_X$, and therefore η contains the mappings to the variables in θ.

We handle the letters in Σ_A in the straightforward way. For every $\sigma \in \Sigma_A$ we add a new state s' to $\mathcal{D}$, if it does not already exist, as follows. For every $\langle q, \eta\rangle \in s$ and $\langle q', \theta\rangle \in Q_U$ such that $\langle q', \theta\rangle \in \rho(\langle q, \eta|_X\rangle, \sigma)$, we add $\langle q', \eta\rangle$ to s'. We define $\delta_D(s, \sigma) = s'$. Notice that by the definition of ρ we have that $\theta = \eta|_X$, and therefore η contains the mappings to the variables in θ.

In order for the result to be full, we add a set of rejecting states $\{q_{rej}\} \times 2^Z$. For a state $s \in Q_D$, let $Z_s = \bigcup_{\langle q, \eta\rangle \in s} \eta|_Z$, and let $Z's$ be the set of variables in Z_s that do not exit s. we add to δ_D transitions from s to $\langle q_{rej}, Z'_s \cup \{y\}\rangle$ labeled by every $z \in Z'_s$, by every $\sigma \in \Sigma_A$ that does not exit s, and if there exists no $z \in (Z \setminus Z_s)$ exiting s, then we also add a transition labeled by y. We add a self-loop to every rejecting state $\langle q_{rej}, Z_s \cup \{y\}\rangle$ labeled by the variables in $Z_s \cup \{y\}$ and the letters in Σ_A.

Finally, the set of accepting states in $\mathcal{D}$ is $\{s | \langle q, \eta\rangle \in s, \langle q, \eta|_X\rangle \in F \times 2^X\}$.

In order to prove the correctness of the determinization procedure, we have the following lemmas.

Lemma 3. *The VFA $\mathcal{D}$ constructed by the determinization procedure is deterministic.*

Proof: By our definition of δ_D, the pattern automaton D of $\mathcal{D}$ is deterministic. For every state $s \in Q_D$, let $Z_s = \bigcup_{\langle q,\eta\rangle \in s}\{\eta|_Z\}$. We show, by renaming every state $s \in Q_D$ by $\langle s, Z_s\rangle$, that $\mathcal{D}$ is identical to its unwinding.[5]

For the initial state s_0, we have that the initial state is $\langle s_0, \emptyset\rangle$. Let s be a state in D obtained while constructing $\mathcal{D}$. If s introduces a new variable z, then by our definition of δ_D there exists a single state s' such that $\delta(s, z) = s'$ and it holds that $Z_{s'} = Z_s \cup \{z\}$. Also by our definition of δ_D, every other transition from s is labeled by d, where d is either a letter in Σ_A or a variable in Z_s. In both cases, for the state s' for which $\delta(s', d) = s'$ we have that $Z'_s = Z_s$. This exactly matches the definition of the unwinding automaton of $\mathcal{D}$. Therefore, we have that $\mathcal{D}$ is identical to its unwinding.

In addition, we have the following. Apart from transitions to the rejecting sinks there are no y transitions, from every state $s \in Q_D$ there exits exactly one transition labeled by a variable $z \notin Z_s$, every $\sigma \in \Sigma_A$ and every $z \in Z_s$ exits s. Therefore, according to the proof of Theorem 7, we have that $\mathcal{D}$ is deterministic. □

Lemma 4. *The VFA $\mathcal{D}$ is equivalent to $\mathcal{A}$.*

Proof: To prove that $L(\mathcal{U}) \subseteq L(\mathcal{D})$, we prove that for every word $v = v_1 v_2 \dots v_m$ read along U with a run $\langle q_0, \emptyset\rangle, \langle q_1, \theta_1\rangle, \dots, \langle q_m, \theta_m\rangle$ there exists a one to one function $f : X \to Z$ such that the following hold.

- The word $f(v)$ can be read along D with a run $s_1, s_2, \dots, s_m$,
- for every $0 \leq i \leq m$, there exists $\langle t, \eta\rangle \in s_i$ such that $\langle t, \eta|_X\rangle = \langle q_i, \theta_i\rangle$, and
- if $v \in L(U)$ then $f(v) \in L(D)$,

where $f(v)$ denotes the replacement of every letter $v_i \in X$ by $f(v_i)$. Notice that v and $f(v)$ are then witnessing patterns for the same set of words.

To prove that $L(\mathcal{D}) \subseteq L(\mathcal{U})$ we prove that for every word $u = u_1 u_2 \dots u_m$ read along D with a run $s_0, s_1, \dots, s_m$ where $s_m = \{\langle q_1, \eta_1\rangle, \langle q_2, \eta_2\rangle, \dots, \langle q_k, \eta_k\rangle\}$, for every $1 \leq i \leq k$ there exists a one to one function $g_i : Z \to X$ such that the following hold.

- The word $g_i(u)$ can be read along U with a run $\langle t_0, \emptyset\rangle, \langle t_1, \theta_1\rangle, \dots, \langle t_m, \theta_m\rangle$,
- for every $0 \leq i \leq m$, there exists $\langle t, \eta\rangle \in s_i$ such that $\langle t, \eta|_X\rangle = \langle t_i, \theta_i\rangle$,
- it holds that $\langle t_m, \theta_m\rangle = \langle q_i, \eta_i|_X\rangle$, and
- if $u \in L(D)$ then there exists g_i such that $g_i(u) \in L(U)$,

[5] Apart for the rejecting sinks construction, which does not affect the correctness

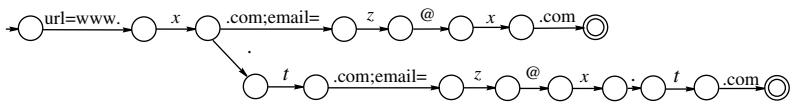

Figure 6: The DVFA equivalent to the VFA from Figure 5

where $g_i(u)$ denotes the replacement of every letter $u_j \in Z$ by $g_i(u_j)$.

Intuitively, this shows that for for every $\langle q_i, \eta_i \rangle \in s_m$, there exists a matching run in $\mathcal{U}$ on u that reaches $\langle q_i, \eta_i \rangle$, and for every $1 \leq i \leq k$, the words $g_i(u)$ and u are witnessing patterns for the same set of words. If s_m is accepting then one of these runs is accepting in $\mathcal{U}$.

The complete proof is given in Appendix A.3 □

Lemma 5. *The determinization procedure is finite and yields a finite DVFA.*

Proof: By the description of the procedure, if $\mathcal{D}$ is finite then the procedure terminates. Notice that by the way $\mathcal{D}$ is defined, and since $\mathcal{U}$ is finite, it suffices to show that Z is finite.

A variable in Z is introduced only from a state s containing a state $\langle q, \eta \rangle$ such that there is a transition from $\langle q, \eta|_X \rangle$ in ρ labeled by a variable $x \notin \eta|_X$. According to the definition of $\mathcal{U}$, in every path $\pi = \langle q_0, \emptyset \rangle, \langle q_1, \theta_1 \rangle, \ldots$ in U we have that $\theta_i \subseteq \theta_{i+1} \subseteq X$ for every $i > 0$. According to the way we defined δ_D, for two states q and s such that $\delta_D(q, z) = s$ for some $z \in Z$ we have that for every $\langle s_i, \beta_i \rangle \in s$ there exists some $\langle q_j, \eta_j \rangle \in q$ such that $\eta_j|_X \subseteq \beta_i|_X$. Therefore, in every path $\pi = s_0, s_1, \ldots$ in D there exists a state s_i such that from s_i on no new variables are introduced. Therefore, every path in $\mathcal{D}$ introduces a finite set of new variables, and hence contains a finite set of states. Since the number of transitions from every state in Q_D is finite, we have that $\mathcal{D}$ is finite. □

Theorem 16. *Let $\mathcal{A}$ be a VFA that has no transitions labeled by the free variable in its pattern automaton. Then there exists a DVFA $\mathcal{D}$ such that $\mathcal{D} \equiv \mathcal{A}$.*

Proof: Theorem 16 follows from Lemmas 3, 4 and 5. □

We show the result of applying the algorithm on the VFA described in Figure 5. For clarity, we do not include in the figure the transition to the rejecting sinks.

7. Variable Büchi Automata

In [6], Büchi extended NFAs to nondeterministic Büchi automata, which run on infinite words. The similarity between VFAs and NFAs enables us to extend VFAs to nondeterministic variable Büchi automata (VBA, for short). Formally, a VBA is $\mathcal{A} = \langle \Sigma, A \rangle$, where A is a nondeterministic Büchi automaton (NBA). Thus, a run of the pattern automaton A is accepting iff it visits the set of accepting states infinitely often. Similar straightforward extensions can be

described for additional acceptance conditions for infinite words. As we specify below, the properties and decision procedures for VFAs generalize to VBA in the expected way, demonstrating the robustness of the VFA formalism.

We start with closure properties. The union construction for VBA is identical to the union construction for VFA. The intersection construction for NBAs involves two copies of the product automaton. Recall that the intersection construction for VFAs involves several copies of the product automaton. Combining the two constructions, we construct the intersection of two VBAs by taking two copies of these several copies. This guarantees that the run has infinitely many visits in accepting states of both VBAs, and that the conflicts between assignments are taken care of. Therefore, we have the following.

Theorem 17. *VBA and DVBA are closed under union and intersection.*

As with VFAs, VBAs are not closed under complementation. Recall that a DVFA can be complemented by complementing its pattern automaton. Since deterministic Büchi automata are not closed under complementation, so are DVBA. However, a deterministic Büchi automata can be translated to an NBA by taking two copies of the given automaton, the second of which contains only the rejecting states. An accepting run continues along the first copy for a finite number of steps and nondeterministically continues on to the second copy. A similar construction is used to complement a DVFA to a VBA.

Theorem 18. *VBAs and DVBAs are not closed under complementation. A DVBA can be complemented to a VBA.*

As for the various decision problems, the complexities and reductions of VFAs all apply, with minor modifications.

Theorem 19.

- *The nonemptiness problem for VBA and DVBA is NL-complete.*
- *The membership problem for VBA is NP-complete and for DVBA is in PTIME.*
- *The containment problem for VBA is undecidable and for DVBA is in co-NP.*
- *Deciding whether a given VBA is a DVBA is NL-complete.*

References

[1] Mikoaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and xml reasoning. *J. ACM*, 56(3):1–48, 2009.

[2] Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *LICS*, pages 7–16. IEEE Computer Society, 2006.

[3] A. Bouajjani, P. Habermehl, and R.: Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 2003.

[4] Ahmed Bouajjani, Peter Habermehl, Yan Jurski, and Mihaela Sighireanu. Rewriting systems with data. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *FCT*, volume 4639 of *LNCS*, pages 1–22. Springer, 2007.

[5] Marco Brambilla, Stefano Ceri, Sara Comai, Piero Fraternali, and Ioana Manolescu. Specification and design of workflow-driven hypertexts. *J. Web Eng.*, 1(2):163–182, 2003.

[6] J.: Büchi. On a decision method in restricted second order arithmetic. In *Int. Congress on Logic, Method, and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.

[7] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[8] S. Demri, R. Lazic, and D.: Nowak. On the freeze quantifier in constraint ltl: Decidability and complexity. *Information and Computation*, 07:2–24, 2007.

[9] O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *The 4th International Conference on Language and Automata Theory and Applications (LATA 2010)*, LNCS, pages 561–572. Springer, 2010.

[10] M. Kaminski and D.: Zeitlin. Extending finite-memory automata with non-deterministic reassignment. In Csuhaj-Varjú and Z. E., Ézik, editors, *AFL*, pages 195–207. In eds.:, 2008.

[11] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[12] Nicolas Markey and Ph. Schnoebelen. Model checking a path. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR*, volume 2761 of *LNCS*, pages 248–262. Springer, 2003.

[13] Frank Neven, Thomas Schwentick, and Victor Vianu. Towards regular languages over infinite alphabets. In *MFCS '01*, pages 560–572, London, UK, 2001. Springer-Verlag.

[14] Y. Shemesh and N.: Francez. Finite-state unification automata and relational languages. *Information and Computation*, 114:192–213, 1994.

[15] T.: Tan. *Pebble Automata for Data Languages: Separation, Decidability, and Undecidability*. PhD thesis, Technion - Computer Science Department, 2009.

[16] Victor Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT '09*, pages 1–13. ACM, 2009.

Appendix A. Proofs

Appendix A.1. Proof of Theorem 8

Formally, the pattern automaton U of $\mathcal{U}$ is $\langle(\Sigma_1 \cup \Sigma_2) \cup X \cup \{y\}, Q, q_0, \delta, F\rangle$ where

- $X = ((X_1 \cup \{y_1\}) \times (X_2 \cup \{y_2\})) \setminus \{\langle y_1, y_2\rangle\}$,
- $y = \langle y_1, y_2\rangle$,
- $Q = (Q_1 \times Q_2) \times 2^{\Gamma_1 \times \Gamma_2}$,
- $q_0 = \langle q_0^1, q_0^2, \emptyset\rangle$,
- $F = (F_1 \times F_2) \times 2^{\Gamma_1 \times \Gamma_2}$, and
- The transition function δ is as follows.

  For a set β in $2^{\Gamma_1 \times \Gamma_2}$, we define $\beta|_1 = \{z_1 | \exists z_2 . \langle z_1, z_2\rangle \in \beta\}$ and $\beta|_2 = \{z_2 | \exists z_1 . \langle z_1, z_2\rangle \in \beta\}$.

  For a letter $a \in \Sigma$, we define $\delta(\langle\langle s, \theta\rangle, \langle t, \eta\rangle, \beta\rangle, a) = \langle\langle s', \theta'\rangle, \langle t', \eta'\rangle, \beta'\rangle$ if

  - $a \in (\Sigma_1 \cap \Sigma_2)$ and $\delta_1(\langle s, \theta\rangle, a) = \langle s', \theta'\rangle$ and $\delta_2(\langle t, \eta\rangle, a) = \langle t', \eta'\rangle$ and $\beta' = \beta$, or
  - $\sigma \in (\Sigma_1 \setminus \Sigma_2)$ and $\delta_1(\langle s, \theta\rangle, a) = \langle s', \theta'\rangle$ and there exists a variable $z_2 \notin \eta$ such that $\delta_2(\langle t, \eta\rangle, z_2) = \langle t', \eta'\rangle$, and $\sigma \notin \beta|_1$, and $\beta' = \beta \cup \{\langle a, z_2\rangle\}$, or
  - $a \in (\Sigma_1 \setminus \Sigma_2)$ and $\delta_1(\langle s, \theta\rangle, a) = \langle s', \theta'\rangle$ and there exists $z_2 \in (X_2 \cup \{y_2\})$ such that $\delta_2(\langle t, \eta\rangle, z_2) = \langle t', \eta'\rangle$, and $\langle a, z_2\rangle \in \beta$, and $\beta' = \beta$.

  We define transitions for $\sigma \in (\Sigma_2 \setminus \Sigma_1)$ by the symmetric conditions.

  For a variable $z = \langle z_1, z_2\rangle$ we define $\delta(\langle\langle s, \theta\rangle, \langle t, \eta\rangle, \beta\rangle, z) = \langle\langle s', \theta'\rangle, \langle t', \eta'\rangle, \beta'\rangle$ if

  - $\delta_1(\langle s, \theta\rangle, z_1) = \langle s', \theta'\rangle$ and $\delta_2(\langle t, \eta\rangle, z_2) = \langle t', \eta'\rangle$, if $z_1 \in X_1$ then $z_1 \notin \theta$, and if $z_2 \in X_2$ then $z_2 \notin \eta$ and $\beta' = \beta \cup \{\langle z_1, z_2\rangle\}$, or
  - $\delta_1(\langle s, \theta\rangle, z_1) = \langle s', \theta'\rangle$ and $\delta_2(\langle t, \eta\rangle, z_2) = \langle t', \eta'\rangle$, and $\langle z_1, z_2\rangle \in \beta$ and $\beta = \beta'$,

To prove that $\mathcal{U}$ is deterministic, we first show that it is identical to its unwinding. Note that since U_1 and U_2 are deterministic and by the way we defined δ, we have that U is deterministic. For a state $q = \langle t_1, t_2, \beta\rangle$ in Q, we define $\theta(q) = \beta \cap ((X_1 \cup \{y_1\}) \times (X_2 \cup \{y_2\}))$. For the initial state q_0 we have that $\theta(q_0) = \emptyset$. Let q and q' be two states in Q. If $\delta(q, \sigma) = q'$ for a letter $\sigma \in \Sigma$, then by the way we defined δ, we have that $\theta(q) = \theta(q')$. If $\delta(q, z) = q'$ for a variable

$z \in X \cup \{y\}$, then by the way we defined δ, we have that $\theta(q') = \theta(q) \cup \{z\}$. Therefore, renaming every state q by $\langle q, \theta(q)\rangle$ results in the unwinding of $\mathcal{U}$.

We now show that the rest of the conditions of Remark 1 hold.

Consider a state $q = \langle\langle s_1, \theta_1\rangle, \langle s_2, \theta_2\rangle, \beta\rangle$ in Q. Since $\mathcal{U}_1$ and $\mathcal{U}_2$ are both deterministic, we have that from $\langle s_1, \theta_1\rangle$ there exits at most one variable not in θ_1 and from $\langle s_2, \theta_2\rangle$ there exits at most one new variable not in θ_2. Therefore, by the way we defined δ, there exits at most one variable from q not in $\theta(q)$. In addition, if y_1 exits $\langle s_1, \theta_1\rangle$ then no other variable not in θ_1 exits it, and if y_2 exists from $\langle s_2, \theta_2\rangle$ then no other variable not in θ_2 exits it. Therefore, we have that if $\langle y_1, y_2\rangle$ exits q then no other variable not in $\theta(q)$ exits it.

We show that every constant letter in $\Sigma_1 \cup \Sigma_2$ exits q. Every constant letter of $\mathcal{A}_1$ exits $\langle s_1, \theta_1\rangle$ and every constant letter of $\mathcal{A}_2$ exits $\langle s_2, \theta_2\rangle$. By the definition of δ, every mutual constant letter exits q. Either a new variable not in θ_1 (resp. θ_2) or y_1 (resp. y_2) exit $\langle s_1, \theta_1\rangle$ (resp. $\langle s_2, \theta_2\rangle$), and δ matches it with every constant letter in $\Sigma_2 \setminus \Sigma_1$ (resp. $\Sigma_1 \setminus \Sigma_2$). In both cases, these constants exit q.

Let $\langle z_1, z_2\rangle$ be a variable in $\theta(q)$. By the definition of δ, we have that $z_1 \in \theta_1$ and $z_2 \in \theta_2$. Every variable in θ_1 exits $\langle s_1, \theta_1\rangle$ and every variable in θ_2 exits $\langle s_2, \theta_2\rangle$. Therefore, by the definition of δ, we have that $\langle z_1, z_2\rangle$ exits q.

Consequently, we have that $\mathcal{U}$ is deterministic.

We now show that $L(\mathcal{U}) \subseteq L(\mathcal{U}_1) \cap L(\mathcal{U}_2)$. Let $w = w_1 w_2 \cdots w_m$ be a word in $L(\mathcal{U})$, and let $v = v_1 v_2 \cdots v_m$ be the witnessing pattern for w, with a run $\langle s_0, t_0, \emptyset\rangle, \langle s_1, t_1, \beta_1\rangle, \ldots \langle s_m, t_m, \beta_m\rangle$ in U. We construct witnessing patterns v^1 and v^2 for w in $\mathcal{U}_1$ and $\mathcal{U}_2$, respectively, as follows.

For every $v_i \in (\Sigma_1 \cap \Sigma_2)$, by the way we defined δ, we have that $\delta_1(s_{i-1}, v_i) = s_i$ and $\delta_2(t_{i-1}, v_i) = t_i$. Accordingly, we define $v_i^1 = v_i^2 = v_i (= w_i)$.

For $v_i = \sigma_1 \in (\Sigma_1 \setminus \Sigma_2)$, let v_j be the first occurence of σ_1 in v. Then, by the way we defined δ, we have that $\delta_2(t_{i-1}, z_2) = t_i$ for some variable z_2 in $\mathcal{U}_2$, and that $\{\langle \sigma, z_2\rangle\} = \beta_j \setminus \beta_{j-1}$ Accordingly, we define $v_i^1 = \sigma_1$ and define $v_i^2 = z_2$. Symmetrically, for $v_i = \sigma_2 \in (\Sigma_2 \setminus \Sigma_1)$, we define $v_i^2 = \sigma_2$ and $v_i^1 = z_1$ for the appropriate z_1 in $\mathcal{U}_1$. In all cases, we have that the variable z_1 (resp. z_2) can indeed be assigned σ_2 (resp. σ_1).

For every $v_i = \langle z_1, z_2\rangle$ in $X \cup \{y\}$, by the way we defined δ, we have that $\delta_1(s_{i-1}, z_1) = s_i$ and $\delta_2(t_{i-1}, z_2) = t_i$. Accordingly, we define $v_i^1 = z_1$ and $v_i^2 = z_2$, and since by the way we defined δ we have that if $z_1 \neq y_1$ then z_1 is matched only with z_2 and vise versa, both z_1 and z_2 can be assigned the same letter w_i.

By the way we defined F we have that the (only) runs on v^1 and v^2 are $s_0, s_1, \ldots, s_m$ and $t_0, t_1, \ldots, t_m$ in $\mathcal{U}_1$ and $\mathcal{U}_2$, respectively, are accepting. In addition, we have that v^1 and v^2 are both witnessing patterns for w. Therefore, we have that $w \in L(\mathcal{U}_1) \cap L(\mathcal{U}_2)$.

Finally, we show that $L(\mathcal{U}_1) \cap L(\mathcal{U}_2) \subseteq L(\mathcal{U})$. Let $w = w_1 w_2 \cdots w_m$ be a word in $L(\mathcal{U}_1) \cap L(\mathcal{U}_2)$, and let $v^1 = v_1^1 v_2^1 \cdots v_m^1$ and $v^2 = v_1^2 v_2^2 \cdots v_m^2$ be the witnessing patterns for w in $\mathcal{U}_1$ and $\mathcal{U}_2$, respectively. Let $\langle s_0, \emptyset\rangle, \langle s_1, \theta_1\rangle, \ldots \langle s_m, \theta_m\rangle$ be the run of U_1 on v^1, and let $\langle t_0, \emptyset\rangle, \langle t_1, \eta_1\rangle, \ldots, \langle t_m, \eta_m\rangle$ be the run of U_2 on v^2. We construct a witnessing pattern v for w in $\mathcal{U}$ with a run $r =$

$\langle\langle s_0, \emptyset\rangle, \langle t_0\emptyset\rangle, \emptyset\rangle, \langle\langle s_1, \theta_1\rangle, \langle t_1, \eta_1\rangle, \beta_1\rangle, \ldots \langle\langle s_m, \theta_m\rangle, \langle t_m, \eta_m\rangle, \beta_m\rangle$ in U such that $\beta_i|_1 \cap X_1 = \theta_i$ and $\beta_i|_2 \cap X_2 = \eta_i$ inductively as follows.

If $v_1^1 \in (\Sigma_1 \cap \Sigma_2)$ then so is v_1^2, and so we have that $v_1^1 = v_1^2$, and accordingly, we define $v_1 = v_1^1$. By the definition of δ, we have that $\delta(\langle\langle s_0, \emptyset\rangle, \langle t_0, \emptyset\rangle, \emptyset\rangle, v_1^1) = \langle\langle s_1, \emptyset\rangle, \langle t_1, \emptyset\rangle, \emptyset\rangle$.

If $v_1^1 \in (\Sigma_1 \setminus \Sigma_2)$ then since both v^1 and v^2 are witnessing patterns for w, we have that v_1^2 is a variable z. We define $v_1 = v_1^1$, and by the way we defined δ we have that $\delta(\langle\langle s_0, \emptyset\rangle, \langle t_0, \emptyset\rangle, \emptyset\rangle, v_1^1) = \langle\langle s_1, \emptyset\rangle, \langle t_1, \eta_1\rangle, \{\langle v_1^1, z\rangle\}\rangle$. Symmetrically, if $v_1^2 \in (\Sigma_2 \setminus \Sigma_1)$ then we have that v_1^1 is a variable z. We define $v_1 = v_1^2$, and by the way we defined δ we have that $\delta(\langle\langle s_0, \emptyset\rangle, \langle t_0, \emptyset\rangle, \emptyset\rangle, v_1^2) = \langle\langle s_1, \emptyset\rangle, \langle t_1, \eta_1\rangle, \{\langle z, v_1^2\rangle\}\rangle$.

If $v_1^1 \in (X_1 \cup \{y_1\})$ and $v_1^2 \in (X_2 \cup \{y_2\})$ then we define $v_1 = \langle v_1^1, v_1^2\rangle$, and by the definition of δ we have that
$\delta(\langle\langle s_0, \emptyset\rangle, \langle t_0, \emptyset\rangle, \emptyset\rangle, \langle v_1^1, v_1^2\rangle) = \langle\langle s_1, \theta_1\rangle, \langle t_1, \eta_1\rangle, \{\langle v_1^1, v_1^2\rangle\}\rangle$.

Let $v_1 v_2 \ldots v_{m-1}$ be the word obtained according to the induction hypothesis, and let $\langle\langle s_{m-1}, \theta_{m-1}\rangle, \langle t_{m-1}, \theta_{m-1}\rangle, \beta_{m-1}\rangle$ be the state the run of U on $v_1 v_2 \ldots v_{m-1}$ reaches. In a similar manner to the base case for $v_m^1 \in (\Sigma_1 \cap \Sigma_2)$, we define $v_m = v_m^1$, and the final step of the run r is $r_m = \langle\langle s_m, \theta_m\rangle, \langle t_m, \eta_m\rangle, \beta_{m-1}\rangle$.

If $v_m^1 \in (\Sigma_1 \setminus \Sigma_2)$, we have that v_m^2 is a variable z. We define $v^m = v_m^1$. If $\langle v_m^1, z\rangle \in \beta_{m-1}$ then the final step of the run r is $r_m = \langle\langle s_m, \theta_m\rangle, \langle t_m \eta_m\rangle, \beta_{m-1}\rangle$.

If $\langle v_m^1, z\rangle \notin \beta_{m-1}$ then we claim that $v_m^1 \notin \beta_{m-1}|_1$, that is, v_m^1 has not been matched with a variable up to this point. To see this, assume by way of contradiction that $v_m^1 \in \beta_{m-1}|_1$. Let v_i^1 be the first occurence of the letter v_m^1 in v^1. Then $i < m-1$, and v_i^2 is a variable z'. By the definition of δ and since r can be constructed for $m-1$ steps, we have that $\langle v_m^1, z'\rangle \in \beta_{m-1}$. Since $\langle v_m^1, z\rangle \notin \beta_{m-1}$, we have that $z' \neq z$. Both v^1 and v^2 are witnessing patterns for w, but z and z' cannot both be assigned v_m^1, a contradiction. Therefore, we have that $v_m^1 \notin \beta_{m-1}|_1$.

If $v_m^2 = y_2$, then the final step of the run r is $r_m = \langle\langle s_m, \theta_m\rangle, \langle t_m \eta_m\rangle, \beta_{m-1} \cup \{\langle v_m^1, y_2\rangle\rangle\}$.

If $v_m^2 \in X_2$, we claim that $v_m^2 \notin \beta_{m-1}|_2$, that is, v_m^2 has not been assigned yet and can be matched with v_m^1. To see this, assume by way of contradiction that $v_m^1 \in \beta_{m-1}|_2$. Let v_i^2 be the first occurence of v_m^2 in v^2. Then $i < m-1$, and v_i^1 is either a variable or a letter. By the way we defined δ and since r can be constructed for $m-1$ steps, we have that $\langle v_i^1, v_m^2\rangle \in \beta_{m-1}$. Since $\langle v_m^1, v_m^2\rangle \notin \beta_{m-1}$, we have that $v_i^1 \neq v_m^1$. Therefore, they are either different letters or that v_i^1 is a variable. In both cases, they cannot be assigned the same letter. However, v_m^2 and v_i^2 must be assigned the same letter, and therefore, since v^1 and v^2 are witnessing patterns for w, this is a contradiction. Then, we have that $v_m^2 \notin \beta_{m-1}|_2$, and therefore $v_m^2 \notin \eta_{m-1}$. Consequently, the final step of the run r is $r_m = \langle\langle s_m, \theta_m\rangle, \langle t_m \eta_m\rangle, \beta_{m-1} \cup \{\langle v_m^1, v_m^2\rangle\rangle\}$.

Similarly, if $v_m^2 \in (\Sigma_2 \setminus \Sigma_1)$, we define $v^m = v_m^2$, and the final step of the run r is constructed according to the various cases.

The final case is for $v_m^1 \in (X_1 \cup \{y_1\})$ and $v_m^2 \in (X_2 \cup \{y_2\})$. We define $v_m = \langle v_m^1, v_m^2\rangle$. If $\langle v_m^1, v_m^2\rangle \in \beta_{m-1}$ then the final step of the run r is $r_m = \langle\langle s_m, \theta_m\rangle, \langle t_m \eta_m\rangle, \beta_{m-1}\rangle$. Otherwise, if $v_m^1 \in X_1$, we claim that

$v_m^1 \notin \beta_{m-1}|_1$. That is, v_m^1 has not been assigned yet. To see this, assume by way of contradiction that $v_m^1 \in \beta_{m-1}|_1$. Let v_i^1 be the first occurence of the variable v_m^1 in v^1. Then $i < m-1$, and v_i^2 is a variable z'. By the way we defined δ and since r can be constructed for $m-1$ steps, we have that $\langle v_m^1, z' \rangle \in \beta_{m-1}$. Since $\langle v_m^1, z \rangle \notin \beta_{m-1}$, we have that $z' \neq z$. Both v^1 and v^2 are witnessing patterns for w, but z and z' cannot both be assigned the same letter as v_m^1, a contradiction.

Similarly, we have that if $v_m^2 \in X_2$, then $v_m^2 \notin \beta_{m-1}|_2$. Accordingly, we have that if $v_m^1 \in X_1$ then $v_m^1 \notin \theta_{m-1}$, and if $v_m^2 \in X_2$, then $v_m^1 \notin \eta_{m-1}$. Therefore, the final step of the run is $r_m = \langle \langle s_m, \theta_m \rangle, \langle t_m, \eta_m \rangle, \beta_{m-1} \cup \{\langle v_m^1, v_m^2 \rangle\} \rangle$.

Since both $\langle s_m, \theta_m \rangle$ and $\langle t_m, \eta_m \rangle$ are accepting, we have that $\langle \langle s_m, \theta_m \rangle, \langle t_m, \eta_m \rangle, \beta_m \rangle$ is accepting, and therefore $v \in L(U)$.

It is left to show that v is a witnessing pattern for w. For the cases where $v_i^1 \in \Sigma$ or $v_i^2 \in \Sigma$ we have that $v_i = w_i$. Otherwise, for every occurence of w_i, we have that $v_i^1 = z_1$ and $v_i^2 = z_2$ for variables $z_1 \in (X_1 \cup \{y_1\})$ and $z_2 \in (X_2 \cup \{y_2\})$, and that $v_i = \langle z_1, z_2 \rangle$. Therefore, every occurence of v_i can be assigned w_i. Consequently, we have that $w \in L(\mathcal{U})$.

Appendix A.2. Proof of Theorem 9

The pattern automaton U of $\mathcal{U}$ is the automaton obtained from the intersection construction for $\mathcal{A}_1$ and $\mathcal{A}_2$ by defining the set of accepting states F as $F = ((F_1 \times Q_2) \cup (Q_1 \times F_2)) \times 2^{\Gamma_1 \times \Gamma_2}$.

To see the correctness of the construction, recall that in the proof of Theorem 8, we show that every word $w \in \Sigma^*$ has a single run r in $\mathcal{U}$. To see that $L(\mathcal{U}) \subseteq L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$, let $\langle p, q, \beta \rangle$ be the last state in r. Then, as we show in the proof of Theorem 8, the word w has runs r_1 and r_2 in $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, such that the last state in r_1 is p and the last state in r_2 is q. Therefore, if r is accepting then either p or q is accepting, and so $w \in L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.

To see that $L(\mathcal{A}_1) \cup L(\mathcal{A}_2) \subseteq L(\mathcal{U})$, let r_1 and r_2 be the runs of $\mathcal{A}_1$ and A_2, respectively, on a word $w \in L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$, and let q_1 and q_2 be the last states in r_1 and r_2, respectively. Then, as we show in the proof of Theorem 8, the word w has a run r in $\mathcal{U}$ such that the last state in r is $\langle q_1, q_2, \beta \rangle$ for some β. Therefore, by the definition of F, we have that $w \in L(\mathcal{U})$.

Appendix A.3. Proof of Lemma 4

To prove that $L(\mathcal{U}) \subseteq L(\mathcal{D})$ we prove that for every word $v = v_1 v_2 \dots v_m$ read along U with a run $\langle q_0, \emptyset \rangle, \langle q_1, \theta_1, \rangle, \dots \langle q_m, \theta_m \rangle$ there exists a one to one function $f : X \to Z$ such that the following hold.

- The word $f(v)$ can be read along D with a run $s_1, s_2, \dots, s_m$,
- For every $0 \leq i \leq m$, there exists $\langle t, \eta \rangle \in s_i$ such that $\langle t, \eta|_X \rangle = \langle q_i, \theta_i \rangle$, and
- If $v \in L(U)$ then $f(v) \in L(D)$,

where $f(v)$ denotes the replacement of every letter $v_i \in X$ by $f(v_i)$.

Notice that v and $f(v)$ are then witnessing patterns for the same set of words.

Let $v = v_1 v_2 \dots v_m$ with a run $\langle q_0, \emptyset\rangle, \langle q_1, \theta_1\rangle, \dots, \langle q_m, \theta_m\rangle$ in U. We prove by induction on m that there exists a suitable function f.

For the base case of $m = 1$, if $v_1 \in \Sigma_A$ then the run in U is $\langle q_0, \emptyset\rangle, \langle q_1, \emptyset\rangle$, and by the way we defined δ_D there exists a state $s_1 \in Q_D$ such that $\delta_D(\{\langle q_0, \emptyset\rangle\}, v_1) = s_1$ and $\langle q_1, \emptyset\rangle \in s_1$. If $\langle q_1, \emptyset\rangle$ is accepting then so is s_1. Since we must have $v_1 = f(v_1)$, we get a suitable function by setting $f = \emptyset$.

If $v_1 \in X$ then the run in U is $\langle q_0, \emptyset\rangle, \langle q_1, \{x\}\rangle$ and the determinization procedure matches v_1 with a new variable z_1. By the way we defined δ_D there exists a state $s_1 \in Q_D$ such that $\delta_D(\{\langle q_0, \emptyset\rangle\}, z_1) = s_1$ and $\langle q_1, \{\langle x, z_1\rangle\}\rangle \in s_1$, and so, in a similar manner, assigning $f(v_1) = z_1$ yields a suitable function.

For the induction step, let h be the function which exists by the induction hypothesis for $v' = v_1 v_2 \dots v_{m-1}$ and let $s_{m-1} \in Q_D$ be the state the run on $h(v')$ reaches in D. Then $\langle q_{m-1}, \theta_{m-1}\rangle = \langle t, \eta|_X\rangle$ for some $\langle t, \eta\rangle \in s_{m-1}$. If $v_m \in \Sigma_A$ then $\theta_{m-1} = \theta_m$. By the way we have defined δ_D, there exists $s_m \in Q_D$ such that $\delta_D(s_{m-1}, v_m) = s_m$ and $\langle q_m, \eta\rangle \in s_m$. If $\langle q_m, \theta_{m-1}\rangle$ is accepting then s_m is accepting, and so $f = h$ yields a suitable function.

If $v_m \in X$ then if $v_m \in \theta_{m-1}$, by the way we have defined D there exists some $z \in Z$ such that $\langle v_m, z\rangle \in \eta$. Also, we have that $\theta_{m-1} = \theta_m$. By the way we have defined δ_D, there exists $s_m \in Q_D$ such that $\delta_D(s_{m-1}, z) = s_m$ and $\langle q_m, \eta\rangle \in s_m$. if $\langle q_m, \theta_{m-1}\rangle$ is accepting then s_m is accepting, and so setting $f = h$ yields a suitable function.

If $v_m \notin \theta_{m-1}$ then there exists no $z \in Z$ such that $\langle v_m, z\rangle \in \eta$, there exists no $i < m$ for which $v_i = v_m$ and we have that $\theta_m = \theta_{m-1} \cup \{v_m\}$. By the way we defined δ_D, during the procedure v_m is matched with a new variable z, and there exists $s_m \in Q_D$ such that $\delta_D(s_{m-1}, z) = s_m$ and $\langle q_m, \eta \cup \{\langle v_m, z\rangle\}\rangle \in s_m$. if $\langle q_m, \theta_m\rangle$ is accepting then s_m is accepting, and so $f = h \cup \{\langle v_m, z\rangle\}$ yields a suitable function.

To prove that $L(\mathcal{D}) \subseteq L(\mathcal{U})$ we prove that for every word $u = u_1 u_2 \dots u_m$ read along D with a run $s_0, s_1, \dots, s_m$ where $s_m = \{\langle q_1, \eta_1\rangle, \langle q_2, \eta_2\rangle, \dots, \langle q_k, \eta_k\rangle\}$, for every $1 \leq i \leq k$ there exists a one to one function $g_i : Z \to X$ such that the following holds.

- The word $g_i(u)$ can be read along U with a run $\langle t_0, \emptyset\rangle, \langle t_1, \theta_1\rangle, \dots, \langle t_m, \theta_m\rangle$,
- For every $0 \leq j \leq m$, there exists $\langle t, \eta\rangle \in s_j$ such that $\langle t, \eta|_X\rangle = \langle t_j, \theta_j\rangle$,
- It holds that $\langle t_m, \theta_m\rangle = \langle q_i, \eta_i|_X\rangle$, and
- If $u \in L(D)$ then there exists g_i such that $g_i(u) \in L(U)$,

where $g_i(u)$ denotes the replacement of every letter $u_j \in Z$ by $g_i(u_j)$.

Intuitively, this shows that for for every $\langle q_i, \eta_i\rangle \in s_m$, there exists a matching run in $\mathcal{U}$ on u that reaches $\langle q_i, \eta_i\rangle$, and for every $1 \leq i \leq k$, the words $g_i(u)$ and u are witnessing patterns for the same set of words. If s_m is accepting then one of these runs is accepting in $\mathcal{U}$.

Let $u = u_1 u_2 \dots u_m$ with a run $s_0, s_1, \dots, s_m$ in D where $s_m = \{\langle q_1, \eta_1 \rangle, \langle q_2, \eta_2 \rangle , \dots, \langle q_k, \eta_k \rangle\}$. We prove by induction on m that there exists a suitable set of functions $\{g_i\}_{1 \leq i \leq k}$.

For the base case of $m = 1$, if $u_1 \in \Sigma_A$ then the run in D is $\{\langle q_0, \emptyset \rangle\}, s_1$, and by the way we defined δ_D it holds that $\langle q_1, \emptyset \rangle \in \rho(\langle q_0, \emptyset \rangle, u_1)$ for some $q_1 \in A$, and since $\mathcal{U}$ is deterministic we have that $s_1 = \{\langle q_1, \emptyset \rangle\}$. If s_1 is accepting then so is $\langle q, \emptyset \rangle$. Since we must have $u_1 = g_1(u_1)$, we get a suitable function by setting $g_1 = \emptyset$.

If $u_1 \in Z$ then by the way we defined $\mathcal{D}$ the run in D is $\{\langle q_0, \emptyset \rangle\}, s_1$ where $s_1 = \{\langle q_1, \eta_1 \rangle, \langle q_2, \eta_2 \rangle, \dots, \langle q_k, \eta_k \rangle\}$ and where every η_i is of the form $\eta_i = \{x_i, u_1\}$ for some $x_i \in X$. If s_1 is accepting then there exists some $\langle q_j, \eta_j \rangle \in s_1$ such that $\langle q_j, \eta_j |_X \rangle$ is accepting, and so setting $g_i(u_1) = x_i$ for $1 \leq i \leq k$ yields a suitable set of functions.

For the induction step, let $s_{m-1} = \{\langle r_1, \eta_1 \rangle, \langle r_2, \eta_2 \rangle, \dots, \langle r_l, \eta_l \rangle\}$ and let $\{h_j\}_{1 \leq j \leq l}$ be the set functions which exists by the induction hypothesis for $u' = u_1 u_2 \dots u_{m-1}$. If $u_m \in \Sigma_A$, since $\delta_D(s_{m-1}, u_m) = s_m$ and by the definition of δ_D, for every $\langle t_i, \eta_i |_X \rangle$ such that $\langle t_i, \eta_i \rangle \in s_m$ there exists $\langle r_j, \eta_j \rangle \in s_{m-1}$ such that $\langle t_i, \eta_i |_X \rangle \in \rho(\langle r_j, \eta_j |_X \rangle, u_m)$. If there exists $\langle t, \eta \rangle \in s_m$ such that $\langle t, \eta |_X \rangle$ is accepting then s_m is accepting, and so we have that setting $g_i = h_j$ yields a suitable set of functions.

If $u_m \in Z$, since $\delta_D(s_{m-1}, u_m) = s_m$ and by the way we have defined δ_D, for every $\langle t_i, \eta_i |_X \rangle$ such that $\langle t_i, \eta_i \rangle \in s_m$ there exists $\langle r_j, \eta_j \rangle \in s_{m-1}$ and $x_j \in X$ such that $\langle t_i, \eta_i |_X \rangle \in \rho(\langle r_j, \eta_j |_X \rangle, x_j)$ where $\langle x_j, u_m \rangle \in \eta_i$. If there exists $\langle t, \eta \rangle \in s_m$ such that $\langle t, \eta |_X \rangle$ is accepting then s_m is accepting, and so we have that setting $g_i = h_j \cup \{\langle u_m, x_j \rangle\}$ yields a suitable set of functions.

Notice that if $\langle x_j, u_m \rangle \in \eta_j$ then $\langle u_m, x_j \rangle \in h_j$. This, since $\langle x_j, u_m \rangle \in \eta_j$ means that $u_m = u_p$ for some $p < m$, and so, according to the induction steps and the definition of δ_D, the variable u_m has already been matched to x_j in h_j. Otherwise, u_m is first introduced from s_{m-1}, and g_i is formed by extending h_j accordingly. In both cases, we have that g_i is well defined.