Variable Automata over Infinite Alphabets

Orna Grumberg[1], Orna Kupferman[2], and Sarai Sheinvald[2]

[1] Department of Computer Science, The Technion, Haifa 32000, Israel
[2] School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel

Abstract. Automated reasoning about systems with infinite domains requires an extension of 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.

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 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 [15, 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*. [3]

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

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

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. We elaborate of the existing formalisms below.

A register automaton [13] 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, [10] forces the content of the registers to be different, [12] adds alternation and two-wayness, and [9] allows the registers to change their content nondeterministically during the run.

A pebble automaton [12] 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, [12] studies alternating and two-way pebble automata, and [14] 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 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.

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 [12]. Pebble automata and most of their variants fail the computability criterion, as apart from weaker models [14], 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 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.

More 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 pre-assigned, 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 unwinded 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 fulfills the three criteria.

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 process 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 5, 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 there exists q' such that $q' \in \delta(q, a)$, we say that a *exits* q. A *run of* A on $w = \sigma_1 \sigma_2 \ldots \sigma_n$ in Γ^* is a sequence of states $r = r_0, r_1, \ldots, r_n$ 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 on which there exists an accepting run of A.

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 $L(A_1) = x \cdot y^* \cdot x$. Consider the language $L' = \{i_1 \cdot i_2 \cdots i_k : k \geq 2, i_1 = i_k, \text{ and } i_j \neq i_1 \text{ for all } 1 < j < k\}$ over the alphabet $\mathbb{N}$; that is, L' contains exactly all words in which the first letter is equal to the last letter, and is different from all other letters. Since $\mathbb{N}$ is infinite, an NFW for it needs infinitely many states and transitions. The idea behind variable automata is to label the transitions of the NFA by both letters from the infinite alphabet and variables that can take values from it. For example, if we refer to x as a bounded variable whose value is fixed once assigned, and refer to y as a free variable, which can take changing values, different from the value assigned to x, then the NFA A_1, when viewed as a variable automaton over $\mathbb{N}$,

recognizes the language L'. Also, if we want to remove the restriction about the letters in the middle being different from the first letter, thus consider $L'' = \{i_1 \cdot i_2 \cdots i_k : k \geq 2 \text{ and } i_1 = i_k\}$, we can label the self loop in A_1 by both x and y.

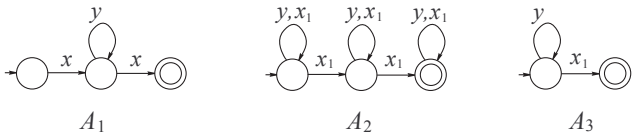

Fig. 1. The pattern automata A_1, A_2, and A_3 for the VFAs $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$

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 $\Gamma_A = \Sigma_A \cup X \cup \{y\}$, 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*. 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 every occurrence 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 these 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.

Comparison with Other Formalisms In terms of expressive power, VFAs are incomparable with FMAs – the register automata of [10], but can be simulated by NFMA [9], which extend FMAs with nondeterministic updates of the registers. 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. VFAs are also incomparable with data automata [2], yet a VFA with no constant letters can be simulated by a data automaton. Intuitively, the transducers of data automata can be used in order to check that the restrictions imposed by the pattern automaton apply.

In the full version, we elaborate more on the relation with the existing formalisms. As detailed there, the examples showing the expressiveness superiority of the existing formalisms are tightly related to their complexity, and we do not find them appealing in practice. For example, it is not surprising that a formalism for which the emptiness problem can be checked in NL (in Theorem 3 we show that emptiness of a VFA can be reduced to emptiness of its pattern automaton) cannot recognize the language of all words in which all letters are different. Data automata can recognize this language since their notion of acceptance involves several runs, on different subwords of the word. Of course, for some applications such an ability is important. VFAs, however, are sufficiently strong to specify many natural properties, and for many applications, we rather give up the expressiveness superiority of the other formalisms for a simple and computationally easy formalism.

3 Properties of VFAs

This section studies closure properties of VFAs and the decidability and complexity of basic problems. We show that VFAs are closed under union and intersection, but 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 NFAs, the membership problem is harder, yet decidable, whereas the universality and containment problems are undecidable.

Theorem 1. *VFAs are closed under union and intersection.*

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

We start with the union construction. The standard construction for NFAs, which guesses whether to follow $\mathcal{A}_1$ or $\mathcal{A}_2$, does not work for VFAs. To see why, note that the range of the variables in a standard union construction would be $\Sigma \setminus (\Sigma_1 \cup \Sigma_2)$. Accordingly, words in $L(\mathcal{A}_1)$ in which variables are assigned values in Σ_2 may be missed, and dually for $L(\mathcal{A}_2)$. We solve this problem by defining the union of $\mathcal{A}_1$ and $\mathcal{A}_2$ as a union of several copies of the underlying VFAs. In each copy, a subset of the variables is taken care of, and transitions labeled by variables from the set are labeled by constants of the other VFA.

We proceed to an intersection construction. Recall that in the product construction for 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 different 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 $z_1 \in \Sigma_1 \cup X_1 \cup \{y_1\}$ and $z_2 \in \Sigma_2 \cup X_2 \cup \{y_2\}$ 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. Formally, we do this by taking several copies of the product construction of the pattern automata, each associated with a relation H that matches the variables and constant letters of $\mathcal{A}_1$ with the variables and constant letters of $\mathcal{A}_2$.

Theorem 2. *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 different. It can be shown that a VFA that recognizes $\tilde{L}$ needs an unbounded number of variables, and therefore does not exist. □

We now turn to study the decidability and complexity of the emptiness, membership and universality problems for VFAs. Checking nonemptiness of existing formalisms is complex and even undecidable. The fact that a bounded variable keeps its value along the run makes the nonemptiness checking of VFAs very simple. In fact, a VFA is nonempty iff its pattern automaton is nonempty. Beyond the straightforward algorithm this induces, it shows that the VFA formalism is indeed very close to the simple formalism of NFAs.

Theorem 3. *The nonemptiness problem for VFA is NL-complete.*

Theorem 4. *The membership problem for VFA is NP-complete.*

The algorithms for the universality and containment problems for the finite-alphabet case rely on the closure of NFAs under complementation, which does not hold for VFAs. Similarly to [12], for register automata, the undecidability of the universality problem for VFA is proved by a reduction from Post's Correspondence Problem. Since we can easily define a universal VFA, undecidability of the containment problem follows too.

Theorem 5. *The universality and containment problems for VFAs are undecidable.*

4 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 2. 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.

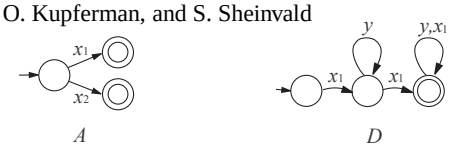

Fig. 2. A nondeterministic VFA whose pattern automaton is deterministic, and a DVFA that accepts all words in which the first letter is repeated at least twice

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.*

Example 3. Consider the VFA $\mathcal{D} = \langle \Sigma, D \rangle$, where D is the DFA appearing in Figure 2. 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 occurrences 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 is easy to see that every word in Σ^* can be read along $\mathcal{D}$. It follows that $\mathcal{D}$ is deterministic.

Although for VFA, unlike NFA, the definition of determinization is semantic, an equivalent syntactic definition does exist, as we show below.

Theorem 6. *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.

- A is nondeterministic, or
- 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 contain x and x', or
- 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 contain x, or
- 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.

Intuitively, the first three conditions check that each word $w \in \Sigma^*$ has at most one run in $\mathcal{A}$. Then, the last condition checks that w has at least one run. In order to implement the above check in NL, we guess the condition that is violated, and check that it is indeed violated. Since NL is closed under complementation, we are done. The lower bound can be shown by a reduction from the reachability problem. □

Note that Theorem 6 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.

We now turn to study the closure properties of DVFAs. Note that closure under union and intersection does not follow from Theorem 1, as here we want to end up with a DVFA and not with a VFA. In order to study the closure properties, we introduce

an *unwinding operator* for VFAs. 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.

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

Intuitively, the states in $\mathcal{U}$ keep track of the set of bounded variables that have been assigned along the paths from the initial state. A run of $\mathcal{A}$ corresponds to a run of $\mathcal{U}$ in which every state is augmented with the set of bounded variables that have appeared earlier in the run. Also, a run of $\mathcal{U}$ corresponds to a run of $\mathcal{A}$ along which the assignments have been accumulated. Therefore, we have that a VFA is equivalent to its unwinding.

We start with union and intersection. The constructions have the construction for DFAs in their basis, applied to the unwinding of the DVFA.

Theorem 7. *DVFA are closed under union and intersection.*

Proof: The constructions for union and intersection both rely on the unwinding of the DVFAs. Since there is a one-to-one correspondence between runs of a VFA and its unwinding, a VFA is deterministic iff its unwinding is deterministic. Let $\mathcal{U}_1$ and $\mathcal{U}_2$ be the unwindings of two DVFAs with pattern automata U_1 and U_2, respectively.

Consider two states q_1 and q_2 in U_1 and U_2, respectively. The intersection construction is based on the product construction of U_1 and U_2. Each state in the unwinding introduces at most one new bounded variable (Theorem 6). If new bounded variables x_1 and x_2 exit q_1 and q_2 respectively, the construction matches x_1 and x_2 together to form a new bounded variable. Similarly for a y_1 transition and a new bounded variable x_2. Transitions labeled by y_1 and y_2 are matched together to form a y transition. The states of the intersection construction keep track of the matchings of bounded variables.

The union of $\mathcal{U}_1$ and $\mathcal{U}_2$ is constructed on top of the intersection construction. Intuitively, a run on the union construction continues along both DVFAs as long as possible. Once it cannot continue along $\mathcal{U}_1$ (w.l.o.g.), it continues along a copy of $\mathcal{U}_2$. As in the proof of Theorem 1, several such copies are taken, in which constants of $\mathcal{U}_1$ are assigned to variables of $\mathcal{U}_2$. □

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 8. *Given a DVFA $\mathcal{A} = \langle \Sigma, A \rangle$ with a set of states Q and a set of accepting states F, let $\tilde{A}$ be the pattern automaton obtained from A by defining its set of accepting states to be $Q \setminus F$, and let $\tilde{\mathcal{A}} = \langle \Sigma, \tilde{A} \rangle$. Then, $L(\tilde{\mathcal{A}}) = \Sigma^* \setminus L(\mathcal{A})$.*

We now study the computability of the DVFA model. We first study the problems of nonemptiness and membership. As argued in the proof of Theorem 3, a VFA is empty iff its pattern automaton is empty. Since the nonemptiness problem in NL-complete also for DFAs, the NL-complete complexity there applies also for DVFAs. For the membership problem, determinism makes the problem easier.

Theorem 9. *The membership problem for DVFA is in PTIME.*

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

Theorem 10. *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 11. *The containment problem for DVFA is in co-NP.*

4.1 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 demonstrates the weakness of DVFA, consider the VFA $\mathcal{A}_2$ of Example 1. In the proof of Theorem 2, we showed that there is no VFA for the complement of $\mathcal{A}_2$. Since DVFAs are closed under complementation, it follows that there is also no DVFA equivalent to $\mathcal{A}_2$.

Theorem 12. *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 returns that no such DVFA exists. We construct from M a Turing machine M' that decides the universality problem for VFA, which, according to Theorem 5, 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. Otherwise, M' returns a DVFA $\mathcal{A}'$ equivalent to $\mathcal{A}$. By Theorem 10, M' can then check $\mathcal{A}'$ for universality. □

However, it turns out that VFA have an expressive determinizable fragment.

Definition 2. *A VFA is* syntactically determinizable *if it has no y transitions.*

For example, consider the syntactically determinizable VFA $\mathcal{A} = \langle \{\mathrm{a}, \ldots, \mathrm{z}\}^*, A \rangle$, appearing in Figure 3. The VFA $\mathcal{A}$ accepts all words of the form

$$\texttt{url=www.}x_1\texttt{.com;email=}z\texttt{@}x_1\texttt{.com} \text{ or}$$
$$\texttt{url=www.}x_2\texttt{.}t\texttt{.com;email=}z\texttt{@}x_2\texttt{.}t\texttt{.com},$$

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`.

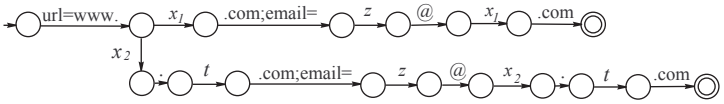

Fig. 3. A syntactically determinizable VFA

Theorem 13. *A syntactically determinizable VFA has an equivalent DVFA.*

The full details of the construction are given in the full paper. Here, we show the result of applying the algorithm on the VFA described in Figure 3. For clarity, we do not include in the figure the transition to the rejecting sinks.

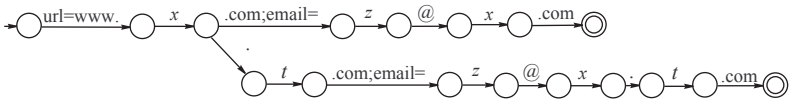

Fig. 4. The DVFA equivalent to the VFA from Figure 3

5 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. Therefore, we have the following.

Theorem 14. *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. Like deterministic Büchi automata, a DVFA can be complemented to a VBA, by translating its pattern automaton to a complementing NBA.

Theorem 15. *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 16.
- *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. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and xml reasoning. J. ACM **56**(3) (2009) 1–48
2. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: LICS, IEEE Computer Society (2006) 7–16
3. Bouajjani, A., Habermehl, P., Mayr, R.: Automatic verification of recursive procedures with one integer parameter. Theoretical Computer Science **295** (2003) 85–106
4. Bouajjani, A., Habermehl, P., Jurski, Y., Sighireanu, M.: Rewriting systems with data. In Csuhaj-Varjú, E., Ésik, Z., eds.: FCT. Volume 4639 of LNCS., Springer (2007) 1–22
5. Brambilla, M., Ceri, S., Comai, S., Fraternali, P., Manolescu, I.: Specification and design of workflow-driven hypertexts. J. Web Eng. **1**(2) (2003) 163–182
6. Büchi, J.: On a decision method in restricted second order arithmetic. In: Int. Congress on Logic, Method, and Philosophy of Science, Stanford University Press (1962) 1–12
7. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)
8. Demri, S., Lazic, R., Nowak, D.: On the freeze quantifier in constraint ltl: Decidability and complexity. Information and Computation **07** (2007) 2–24
9. Kaminski, M., Zeitlin, D.: Extending finite-memory automata with non-deterministic reassignment. In Csuhaj-Varjú, E., Ézik, Z., eds.: AFL, In eds.: (2008) 195–207
10. Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science **134**(2) (1994) 329–363
11. Markey, N., Schnoebelen, P.: Model checking a path. In Amadio, R.M., Lugiez, D., eds.: CONCUR. Volume 2761 of LNCS., Springer (2003) 248–262
12. Neven, F., Schwentick, T., Vianu, V.: Towards regular languages over infinite alphabets. In: MFCS '01, London, UK, Springer-Verlag (2001) 560–572
13. Shemesh, Y., Francez, N.: Finite-state unification automata and relational languages. Information and Computation **114** (1994) 192–213
14. Tan, T.: Pebble Automata for Data Languages: Separation, Decidability, and Undecidability. PhD thesis, Technion - Computer Science Department (2009)
15. Vianu, V.: Automatic verification of database-driven systems: a new frontier. In: ICDT '09, ACM (2009) 1–13