An Automata-Theoretic Approach to Modeling Systems and Specifications over Infinite Data

Hadar Frenkel[1(✉)], Orna Grumberg[1], and Sarai Sheinvald[2]

[1] Department of Computer Science, The Technion, Haifa, Israel
hfrenkel@cs.technion.ac.il

[2] Department of Software Engineering, ORT Braude Academic College, Karmiel, Israel

Abstract. Data-parameterized systems model finite state systems over an infinite data domain. VLTL is an extension of LTL that uses variables in order to specify properties of computations over infinite data, and as such VLTL is suitable for specifying properties of data-parameterized systems. We present *Alternating Variable Büchi Word Automata* (AVBWs), a new model of automata over infinite alphabets, capable of modeling a significant fragment of VLTL. While alternating and non-deterministic Büchi automata over finite alphabets have the same expressive power, we show that this is not the case for infinite data domains, as we prove that AVBWs are strictly stronger than the previously defined Non-deterministic Variable Büchi Word Automata (NVBWs). However, while the emptiness problem is easy for NVBWs, it is undecidable for AVBWs. We present an algorithm for translating AVBWs to NVBWs in cases where such a translation is possible. Additionally, we characterize the structure of AVBWs that can be translated to NVBWs with our algorithm, and identify fragments of VLTL for which a direct NVBW construction exists. Since the emptiness problem is crucial in the automata-theoretic approach to model checking, our results give rise to a model-checking algorithm for a rich fragment of VLTL and systems over infinite data domains.

1 Introduction

Infinite data domains become increasingly relevant and wide-spread in real-life systems, and are integral in communication systems, e-commerce systems, large databases and more. Systems over infinite data domains were studied in several contexts and especially in the context of datalog systems [4] and XML documents [5,7], that are the standard of web documents.

Temporal logic, particularly LTL, is widely used for specifying properties of ongoing systems. However, LTL is unable to specify computations that handle infinite data. Consider, for example, a system of processes and a scheduler. If the set of processes is finite and known in advance, we can express and verify properties such as "every process is eventually active". However, if the system is dynamic, in which new processes can log in and out, and the total number of processes is unbounded, LTL is unable to express such a property.

C. Barrett et al. (Eds.): NFM 2017, LNCS 10227, pp. 1–18, 2017.
DOI: 10.1007/978-3-319-57288-8_1

VLTL (LTL with variables) [11] extends LTL with variables that range over an infinite domain, making it a natural logic for specifying ongoing systems over infinite data domains. For the example above, a VLTL formula can be $\varphi_1 = \forall x : \ \mathsf{G}\,(\mathit{loggedIn}(x) \rightarrow \mathsf{F}\,(\mathit{active}(x)))$, where x ranges over the process IDs. Thus, the formula specifies that for every process ID, once it is logged in, it will eventually be active. Notice that this formula now specifies this property for an unbounded number of processes. As another example, the formula $\varphi_2 = \mathsf{G}\,\exists x(\mathit{send}(x) \wedge \mathsf{F}\,\mathit{receive}(x))$, where x ranges over the message contents (or message IDs), specifies that in every step of the computation, some message is sent, and this particular message is eventually received. Using variables enables handling infinitely many messages along a single computation.

In the *automata-theoretic approach to model checking* [18,19], both the system and the specification are modeled by automata whose languages match the set of computations of the system and the set of satisfying computations of the formula. Model-checking is then reduced to reasoning about these automata. For ongoing systems, automata over infinite words, particularly nondeterministic and alternating Büchi automata (NBWs and ABWs, respectively) are used [18]. Thus, for ongoing systems with infinite data and VLTL, a similar model is needed, capable of handling infinite alphabets. In [10,11], the authors suggested *nondeterministic variable Büchi word automata* (NVBWs), a model that augments NBWs with variables, and used it to construct a model-checking algorithm for a fragment of VLTL that is limited to $\exists$-quantifiers that appear only at the head of the formula.

The emptiness problem for NVBWs is NLOGSPACE-complete. Since the emptiness problem is crucial for model checking, NVBWs are an attractive model. However, they are quite weak. For example, NVBWs are unable to model the formula φ_2 above.

In this work, we present a new model for VLTL specifications, namely *alternating variable Büchi word automata* (AVBWs). These are an extension of NVBWs, which we prove to be stronger and able to express a much richer fragment of VLTL. Specifically, we show that AVBWs are able to express the entire fragment of $\exists^*$-VLTL, which is a fragment of VLTL with only $\exists$-quantifiers, whose position in the formula is unrestricted.

We now elaborate more on NVBWs and AVBWs. As mentioned, an NVBW $\mathcal{A}$ uses variables that range over an infinite alphabet Γ. A run of $\mathcal{A}$ on a word w assigns values to the variables in a way that matches the letters in w. For example, if a letter $a.8$ occurs in w, then a run of $\mathcal{A}$ may read $a.x$, where x is assigned 8. In addition, the variables may be reset at designated states along the run, and so $a.x$ can be later used for reading another letter $a.5$, provided that x has been reset. Resetting then allows reading an unbounded number of letters using a fixed set of variables. Another component of NVBWs is an inequality set E , that allows restricting variables from being assigned with the same value. Our new model of AVBWs extends NVBWs by adding *alternation*. An alternating automaton may split its run and continue reading the input along several different paths simultaneously, all of which must accept.

There is a well-known translation from LTL to ABW [18]. Thus, AVBWs are a natural candidate for modeling VLTL. Indeed, as we show, AVBWs are able to express all $\exists^*$-VLTL, following a translation that is just as natural as the LTL to ABW translation. Existential quantifiers (anywhere) in the formula are translated to corresponding resets in the automaton. Moreover, unlike the finite alphabet case, in which NBWs and ABWs are equally expressive, in the infinite alphabet case alternation proves to be not only syntactically stronger but also semantically stronger, as we show that AVBWs are more expressive than NVBWs.

As we have noted, our goal is to provide a model which is suitable for a model-checking algorithm for VLTL, and that such a model should be easily tested for emptiness. However, we show that the strength of AVBWs comes with a price, and their emptiness problem is unfortunately undecidable. To keep the advantage of ease of translation of VLTL to AVBWs, as well as the ease of using NVBWs for model-checking purposes, we would then like to translate AVBWs to NVBWs, in cases where such a translation is possible. This allows us to enjoy the benefit of both models, and gives rise to a model-checking algorithm that is able to handle a richer fragment of VLTL than the one previously studied.

We present such a translation algorithm, inspired by the construction of [14]. As noted, such a translation is not always possible. Moreover, we show that there is no algorithm that is both sound and complete, even if we restrict completeness to require returning "no translation possible". Our algorithm is then sound but incomplete, and we present an example for which it will not halt. However, we give a characterization for AVBWs for which our algorithm does halt, relying on the graphical structure of the underlying automaton. The essence of the characterization is that translatable AVBWs do not have a cycle that contains a reset action which leads to an accepting state. Consider once again $\varphi_2 = \mathsf{G}\,\exists x(\mathit{send}.x \wedge \mathsf{F}\,\mathit{receive}.x)$. Here, we keep sending messages that must arrive eventually. However, there is no bound on when they will arrive. Since this is a global requirement, there must be some cycle that verifies it, and such cycles are exactly the ones that prevent the run of the translation algorithm from halting.

The importance of our algorithm and structural characterization is a twofold: (1) given an AVBW $\mathcal{A}$, one does not need to know the semantics of $\mathcal{A}$ in order to know if it is translatable, and to automatically translate $\mathcal{A}$ to an equivalent NVBW; and (2) Given a general $\exists^*$-VLTL formula, one can easily construct an equivalent AVBW $\mathcal{A}$, use our characterization to check whether it is translatable, and continue with the NVBW that our translation outputs.

In addition to the results above, we also study fragments of $\exists^*$-VLTL that have a direct construction to NVBWs, making them an "easy" case for modeling and model checking.

Related Work. Other models of automata over infinite alphabets have been defined and studied. In [13] the authors define *register automata over infinite alphabets*, and study their decidability properties. [16] use register automata as well as *pebble automata* to reason about first order logic and monadic second order logic, and to describe XML documents. [3] limit the number of variables and use extended first order logic to reason about both XML and some verification properties. In [4] the authors model infinite state systems as well as infinite data domains, in order to express some extension of monadic first order logic. Our model is closer to finite automata over infinite words than the models above, making it easier to understand. Moreover, due to their similarity to ABWs, we were able to construct a natural translation of $\exists^*$-VLTL to AVBWs, inspired by [18]. We then translate AVBWs to NVBWs. Our construction is consistent with [14] which provides an algorithm for translating ABWs to NBWs. However, in our case additional manipulations are needed in order to handle the variables and track their possible assignments.

The notion of LTL over infinite data domains was studied also in the field of runtime verification (RV) [1,2,8]. Specifically, in [1], the authors suggest a model of quantified automata with variables, in order to capture traces of computations with different data values. The purpose in RV is to check whether a single given trace satisfies the specification. Moreover, the traces under inspection are finite traces. This comes into play in [1] where the authors use the specific data values that appear on such a trace in order to evaluate satisfiability. In [2] the authors suggest a 3-valued semantics in order to capture the uncertainty derived from the fact that traces are finite. Our work approaches infinite data domains in a different manner. Since we want to capture both infinite data domains and infinite traces, we need a much more expressive model, and this is where AVBWs come into play.

2 Preliminaries

Given a finite set of directions D, a *D-tree* T is a set $T \subseteq D^*$. The root of T is the empty word ϵ. A node x of T is a word over D that describes the path from the root of T to x. That is, for a word $d = d_1, d_2, \cdots, d_n$ there is a path in the tree $\pi = \epsilon, d_1, d_1 d_2, \cdots, d_1 d_2 \cdots d_n$ such that every word $d' d_i$ is a successor of the previous word d'. For a word $w \cdot x \in D^*$ where $w \in D^*$ and $x \in D$, if $w \cdot x \in T$ then $w \in T$, i.e. the tree is prefix closed. A successor of a node $w \in T$ is of the form $w \cdot x$ for $x \in D$.

Given a set L, an *L-labeled D-tree* is a pair $\langle T, f \rangle$ where T is a D-tree, and $f : T \to L$ is a labeling function that labels each node in T by an element of L.

A *non-deterministic Büchi automaton* over infinite words (NBW) [6] is a tuple $\mathcal{B} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ where Σ is a finite alphabet; Q is a finite set of states; $q_0 \in Q$ is the initial state; $\alpha \subseteq Q$ is a set of accepting states; and $\delta : Q \times \Sigma \to 2^Q$ is the transition function. For a word $\rho \in \Sigma^\omega$, we denote by ρ_i the letter of ρ in position i.

A *run* of $\mathcal{B}$ on a word $\rho \in \Sigma^\omega$ is an infinite sequence of states $q_0, q_1, q_2, \cdots \in Q^\omega$, that is consistent with δ, i.e., q_0 is the initial state and $\forall i > 0 : q_i \in \delta(q_{i-1}, \rho_i)$. A run of $\mathcal{B}$ is *accepting* if it visits some state of α infinitely often. We say that $\mathcal{B}$ *accepts* a word ρ if there exists an accepting run of $\mathcal{B}$ on ρ. The *language* of $\mathcal{B}$, denoted $\mathcal{L}(\mathcal{B})$, is the set of words accepted by $\mathcal{B}$.

An *alternating Büchi automaton* over infinite words (ABW) [15] is a tuple $\mathcal{B}_A = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ where Σ, Q, q_0 and α are as in NBW. The transition relation is $\delta : Q \times \Sigma \to B^+(Q)$, where $B^+(Q)$ is the set of positive boolean formulas over the set of states, i.e. formulas that include only the boolean operators $\wedge$ and $\vee$[1]. For example, if $\delta(q, a) = (q_1 \wedge q_2) \vee q_3$, then, by reading a from q, the ABW $\mathcal{B}_A$ moves to either both q_1 and q_2, or to q_3. We assume that δ is given in a disjunctive normal form (DNF).

A *run* of an ABW is a Q-labeled Q-tree. Disjunctions are equivalent to nondeterministic choices, and so every disjunct induces a tree. A conjunction induces a split to two or more successors. For example, $\delta(q, a) = (q_1 \wedge q_2) \vee q_3$ induces two trees. In the first, q has two successors, q_1 and q_2. In the second tree the only successor of q is q_3. A run is *accepting* if every infinite path in the corresponding tree visits a state from α infinitely often, and every finite path ends with *true* . The notions of acceptance and language are as in NBWs.

We say that an automaton (either NBW or ABW) is a *labeled automaton* if its definition also includes a labeling function $\mathscr{L} : Q \to L$ for its states, where L is a set of labels. We use this notion to conveniently define variable automata later on.

We assume that the reader is familiar with the syntax and semantics of LTL.

Variable LTL, or *VLTL*, as defined in [11], extends LTL by augmenting atomic propositions with variables. Let AP be a set of parameterized atomic propositions, let X be a finite set of variables, and let $\bar{x}$ be a vector of variables. Then, the formulas in VLTL are over $AP \times X$, thus allowing the propositions to carry data from an infinite data domain. We inductively define the syntax of VLTL.

- For every $a \in AP$ and $x \in X$ the formulas $a.x$ and $\neg a.x$ are VLTL formulas[2].
- For a VLTL formula $\varphi(\bar{x})$ and $x \in X$, the formulas $\exists x \varphi(\bar{x})$ and $\forall x \varphi(\bar{x})$ are in VLTL.
- If $\varphi_1(\bar{x})$ and $\varphi_2(\bar{x})$ are VLTL formulas, then so are $\varphi_1(\bar{x}) \vee \varphi_2(\bar{x})$; $\varphi_1(\bar{x}) \wedge \varphi_2(\bar{x})$; $\mathsf{X}\, \varphi_(\bar{x})$; $\mathsf{F}\, \varphi_1(\bar{x})$; $\mathsf{G}\, \varphi_1(\bar{x})$; $\varphi_1(\bar{x})\, \mathsf{U}\, \varphi_2(\bar{x})$; and $\varphi_1(\bar{x})\, \mathsf{V}\, \varphi_2(\bar{x})$, where V is the release operator, which is the dual operator of U .

Given an alphabet Γ, an assignment $\theta : X \to \Gamma$, and a word $\rho \in (2^{AP \times \Gamma})^\omega$, we denote $\rho \vDash_\theta \varphi(\bar{x})$ if $\rho \vDash \varphi(\bar{x})_{[\bar{x} \leftarrow \theta(\bar{x})]}$ under the standard semantics of LTL. For example, for $\rho = \{p.1\}^\omega$ it holds that $\rho \vDash_\theta \mathsf{G}\, p.x$ for $\theta(x) = 1$.

We denote $\rho \vDash_\theta \exists x \varphi(\bar{x})$ if there exists an assignment $x \leftarrow d$ to the variable x such that $\rho \vDash_\theta \varphi(\bar{x})_{[x \leftarrow d]}$, where $\vDash_\theta$ is as defined before. We denote $\rho \vDash_\theta \forall x \varphi(\bar{x})$ if for every assignment $x \leftarrow d$ to the variable x, it holds that $\rho \vDash_\theta \varphi(\bar{x})_{[x \leftarrow d]}$.

We say that a formula φ is *closed* if every occurrence of a variable in φ is under the scope of a quantifier. Notice that the satisfaction of closed formulas is independent of specific assignments. For a closed formula φ over $\bar{x}$, we then write $\rho \vDash \varphi(\bar{x})$.

The logic $\exists^*$-VLTL is the set of all closed VLTL formulas in negation normal form (NNF) that only use the $\exists$-quantifier. Note that the $\exists$-quantifier may appear

[1] In particular, the negation operator is not included.

[2] The semantics of $\neg a.x$ is regarding a specific value. I.e., if $x = d$ then $a.d$ does not hold, but $a.d'$ for $d \neq d'$ may hold.

anywhere in the formula. The logic $\exists^*_{pnf}$-VLTL is the set of all $\exists^*$-VLTL formulas in prenex normal form, i.e., $\exists$-quantifiers appear only at the beginning of the formula.

The *language* of a formula φ, denoted $\mathcal{L}(\varphi)$, is the set of computations that satisfy φ.

We now define *non-deterministic variable Büchi automata* over infinite words (NVBWs). Our definition is tailored to model VLTL formulas, and thus is slightly different from the definition in [10]. Specifically, the alphabet consists of subsets of $AP \times X$, where AP is a finite set of parameterized atomic propositions.

An NVBW is a tuple $\mathcal{A} = \langle \mathcal{B}, \Gamma, E \rangle$, where $\mathcal{B} = \langle 2^{AP\times X}, Q, q_0, \delta, \mathit{reset}, \alpha \rangle$ is a labeled NBW, X is a finite set of variables, $\mathit{reset} : Q \to 2^X$ is a labeling function that labels each state q with the set of variables that are reset at q, the set $E \subseteq \{x_i \neq x_j | x_i, x_j \in X\}$ is an inequality set over X, and Γ is an infinite alphabet.

A run of an NVBW $\mathcal{A} = \langle \mathcal{B}, \Gamma, E \rangle$ on a word $\rho \in (2^{AP\times\Gamma})^\omega$, where $\rho = \rho_1\rho_2\cdots$ is a pair $\langle \pi, r \rangle$ where $\pi = (q_0, q_1, q_2, \cdots)$, is an infinite sequence of states, and $r = (r_0, r_1, \cdots)$ is a sequence of mappings $r_i : X \to \Gamma$ such that:

1. There exists a word $z \in (2^{AP\times X})^\omega$ such that $\forall i : r_i(z_i) = \rho_i$ and π is a run of $\mathcal{B}$ on z. We say that z is a *symbolic word* that is consistent on $\langle \pi, r \rangle$ with the *concrete word* ρ.
2. The run respects the reset actions: for every $i \in \mathbb{N}, x \in X$, if $x \notin \mathit{reset}\,(q_i)$ then $r_i(x) = r_{i+1}(x)$.
3. The run respects E : for every $i \in \mathbb{N}$ and for every inequality $(x_m \neq x_l) \in E$ it holds that $r_i(x_l) \neq r_i(x_m)$.

A run $\langle \pi, r \rangle$ on ρ is *accepting* if π is an accepting run of $\mathcal{B}$ on a symbolic word z that corresponds to ρ on $\langle \pi, r \rangle$, i.e. π visits α infinitely often. The notion of acceptance and language are as in NBWs.

Intuitively, a run of an NVBW $\mathcal{A}$ on a word ρ assigns each occurrence of a variable a letter from Γ. A variable can "forget" its value only if a reset action occurred. The inequality set E prevents from certain variables to be assigned with the same value.

We say that an NVBW $\mathcal{A}$ *expresses* a formula φ if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi)$.

Example 1. Consider the concrete word $\rho = \{\mathit{send}.1\}, (\{\mathit{send}.2, \mathit{rec}.1\}, \{\mathit{send}.1, \mathit{rec}.2\})^\omega$. In an NVBW $\mathcal{A}$, a corresponding symbolic word can be $z = \{\mathit{send}.x_1\}, (\{\mathit{send}.x_2, \mathit{rec}.x_1\}, \{\mathit{send}.x_1, \mathit{rec}.x_2\})^\omega$. If $\mathcal{A}$ includes reset actions for x_1 and x_2 in every even state in some path of $\mathcal{A}$, then another concrete word consistent with z can be $\rho' = \{\mathit{send}.1\}, \{\mathit{send}.2, \mathit{rec}.1\}, \{\mathit{send}.3, \mathit{rec}.4\}, \{\mathit{send}.4, \mathit{rec}.3\}, \{\mathit{send}.5, \mathit{rec}.6\}, \cdots$, since the values of x_1 and x_2 can change at every even state.

3 Variable Automata: Non-determinism Vs. Alternation

In Sect. 5 we show that NVBWs are useful for model checking in our setting, since they have good decidability properties. In particular, there is a polynomial construction for intersection of NVBWs, and their emptiness problem is

NLOGSPACE-complete [10]. In Sect. 4 we describe a translation of $\exists^*_{pnf}$-VLTL formulas to NVBWs. We now show that NVBWs are too weak to express all VLTL formulas, or even all $\exists^*$-VLTL formulas. It follows that $\exists^*$-VLTL is strictly more expressive than $\exists^*_{pnf}$-VLTL. Nevertheless, we use NVBWs for model checking at least for some fragments of $\exists^*$-VLTL.

Before discussing the properties of variable automata, we first give some motivation for their definition, as given in Sect. 2. In particular, we give motivation for the *reset* labeling function and for E , the inequity set.

Example 2. We begin with resets. Consider the $\exists^*$-VLTL formula $\varphi_1 = \mathsf{G}\,\exists x(a.x)$. One possible computation satisfying φ_1 is $\rho = a.1, a.2, a.3, \cdots$. No NVBW with a finite number of variables can read ρ, unless some variable is reassigned. The reset action allows these reassignments.

Example 3. To see the necessity of the inequality set E , consider the $\exists^*$-VLTL formula $\varphi_2 = \exists x(G\neg a.x)$. We can use a variable x to store a value that never appears along the computation with a. Imposing inequality restrictions on x with all other variables makes sure that the value assigned to x does not appear along the computation via assignments to other variables. Note that if the logic does not allow negations at all, the inequality set is not needed.

3.1 NVBWs Are Not Expressive Enough for ∃*-VLTL

We first show that NVBWs cannot express every $\exists^*$-VLTL formula.

Lemma 1. *The formula* $\varphi_{\mathsf{G}\,\exists} = \mathsf{G}\exists x(b.x \wedge \mathsf{F}a.x)$ *cannot be expressed by an NVBW.*

Proof. Consider the following word ρ over $AP = \{a, b\}$ and $\Gamma = \mathbb{N}$.

$$\rho = \{a.1, b.1\}, \{a.2, b.2\}, \{a.2, b.3\}, \{a.3, b.4\}, .., \{a.4, b.8\} .., \{a.(k+1), b.2^k\}, \cdots$$

i.e., $b.(i)$ occurs in ρ_i, and $a.(i+1)$ occurs for the first time in ρ_{2^i} and continues until $\rho_{2^{i+1}-1}$.
It is easy to see that ρ satisfies $\varphi_{\mathsf{G}\,\exists}$ since at step t for $x = t$ we have that $b.t$ holds, and at some point in the future, specifically at step 2^{t-1}, the proposition $a.t$ will hold.

Assume, by way of contradiction, that $\mathcal{A}$ is an NVBW with m variables that expresses $\varphi_{\mathsf{G}\,\exists}$. Then over a sub-word with more than m values for b, one variable must be reset and used for two different values. We can then create a different computation in which the value that was “forgotten” never appears with a, thus not satisfying $\varphi_{\mathsf{G}\,\exists}$, but accepted by $\mathcal{A}$, a contradiction. □

Not only $\exists$-quantifiers are problematic for NVBWs. NVBWs cannot handle $\forall$-quantifiers, even in PNF. The proof of the following Lemma is almost identical to the proof of Lemma 1.

Lemma 2. *The formula* $\varphi = \forall x : \mathsf{G}(a.x \rightarrow \mathsf{F}b.x)$ *cannot be expressed by an NVBW.*

3.2 Alternating Variable Büchi Automata

In Sect. 3.1 we have shown that NVBWs are not expressive enough, even when considering only the fragment of $\exists^*$-VLTL. We now introduce *alternating variable Büchi automata* over infinite words (AVBW), and show that they can express all of $\exists^*$-VLTL. We study their expressibility and decidability properties.

Definition 1. *An AVBW is a tuple $\mathcal{A} = \langle \mathcal{B}_A, \Gamma, E \rangle$ where $\mathcal{B}_A = \langle 2^{AP \times X}, Q, q_0, \delta, reset, \alpha \rangle$ is a labeled ABW, X is a finite set of variables, $reset : Q \rightarrow 2^X$ is a labeling function that labels every state q with the set of variables that are reset at q, the set E is an inequality set, and Γ is an infinite alphabet. We only allow words in which a proposition $a.\gamma$ for $\gamma \in \Gamma$ appears at most once in every computation step, i.e., no word can contain both $a.\gamma$ and $a.\gamma'$ for $\gamma \neq \gamma'$ at the same position.*

A *run* of an AVBW $\mathcal{A}$ on a word $\rho \in (2^{AP \times \Gamma})^\omega$ is a pair $\langle T, r \rangle$ where T is a Q-labeled Q-tree and r labels each node t of T by a function $r_t : X \rightarrow \Gamma$ such that:

1. The root of T is labeled with q_0.
2. For each path π on T there exists a symbolic word $z_\pi \in (2^{AP \times X})^\omega$ such that $r_{\pi_i}((z_\pi)_i) = \rho_i$.
3. The run respects δ: for each node $t \in T$ labeled by q of depth i on path π, the successors of t are labeled by $q_1, \cdots, q_t$ iff one of the conjuncts in $\delta(q, (z_\pi)_i)$ is exactly $\bigwedge_{j=1..t} q_j$.
4. The run respects the reset actions: if t' is a child node of t labeled by q and $x \notin reset(q)$, then $r_t(x) = r_{t'}(x)$.
5. The run respects E: for every $(x_i \neq x_j) \in E$ and for every node $t \in T$ it holds that $r_t(x_i) \neq r_t(x_j)$.

Intuitively, much like in NVBWs, the variables in every node in the run tree are assigned values in a way that respects the resets and the inequality set.

A run $\langle T, r \rangle$ on ρ is *accepting* if every infinite path π is labeled infinitely often with states in α. The notion of acceptance and language are as usual. Note that the same variable can be assigned different values on different paths.

Just like ABWs, AVBWs are naturally closed under union and intersection. However, unlike ABWs, they are not closed under complementation. We prove this in Sect. 3.4.

3.3 AVBWs Can Express All of $\exists^*$-VLTL

We now show that AVBWs can express $\exists^*$-VLTL. Together with Lemma 1, we reach the following surprising theorem.

Theorem 1. *AVBWs are strictly more expressive than NVBWs.*

This is in contrast to the finite alphabet case, where there are known algorithms for translating ABWs to NBWs [14].

Theorem 2. *Every $\exists^*$-VLTL φ formula can be expressed by an AVBW $\mathcal{A}_\varphi$.*

We start with an example AVBW for $\varphi_{\mathsf{G}\,\exists} = \mathsf{G}\,\exists x(b.x \wedge \mathsf{F}\, a.x)$ from Lemma 1.

Example 4. Let $\mathcal{A} = \langle \mathcal{B}, \mathbb{N}, \emptyset \rangle$ where $\mathcal{B} = \langle 2^{AP \times \{x_1, x_2, x_3\}}, \{q_0, q_1\}, q_0, \delta, \mathit{reset}, \{q_0\} \rangle$.

- $\mathit{reset}(q_0) = \{x_1, x_2\}$, $\mathit{reset}(q_1) = \{x_2, x_3\}$
- $\delta(q_0, \{b.x_1\}) = \delta(q_0, \{a.x_2, b.x_1\}) = q_0 \wedge q_1$
  $\delta(q_0, \{b.x_1, a.x_1\}) = \mathit{true}$
  $\delta(q_1, \{b.x_2\}) = \delta(q_1, \{a.x_2\}) = \delta(q_1, \{a.x_2, b.x_3\}) = q_1$
  $\delta(q_1, \{a.x_1\}) = \delta(q_1, \{a.x_1, b.x_2\}) = \mathit{true}$

Intuitively, q_0 makes sure that at each step there is some value with which b holds. The run then splits to both q_0 and q_1. The state q_1 waits for a with the same value as was seen in q_0 (since x_1 is not reset along this path, it must be the same value), and uses x_2, x_3 to ignore other values that are attached to a, b. The state q_0 continues to read values of b (which again splits the run), while using x_2 to ignore values assigned to a. See Fig. 1 for a graphic representation of $\mathcal{A}$.

We now proceed to the proof of Theorem 2.

Proof. Let φ be an $\exists^*$-VLTL formula. We present an explicit construction of $\mathcal{A}_\varphi$, based on the construction of [18] and by using resets to handle the $\exists$-quantifiers, and inequalities to handle negations. First, we rename the variables in φ and get an equivalent formula φ', where each existential quantifier bounds a variable with a different name. For example, if $\varphi = \exists x(a.x \,\mathsf{U}\, \exists x(b.x))$ then $\varphi' = \exists x_1(a.x_1 \,\mathsf{U}\, \exists x_2(b.x_2))$. Let $\mathit{sub}(\varphi)$ denote all sub-formulas of φ and let $\mathit{var}(\varphi)$ denote the set of variables that appear in φ.

Let $\mathcal{A}_\varphi = \langle \mathcal{B}, \Gamma, E \rangle$ where $\mathcal{B} = \langle 2^{AP \times X}, Q, q_0 = \varphi', \delta, \mathit{reset}, \alpha \rangle$ and where

- $X = \mathit{var}(\varphi') \cup \{x_p | p \in AP\}$

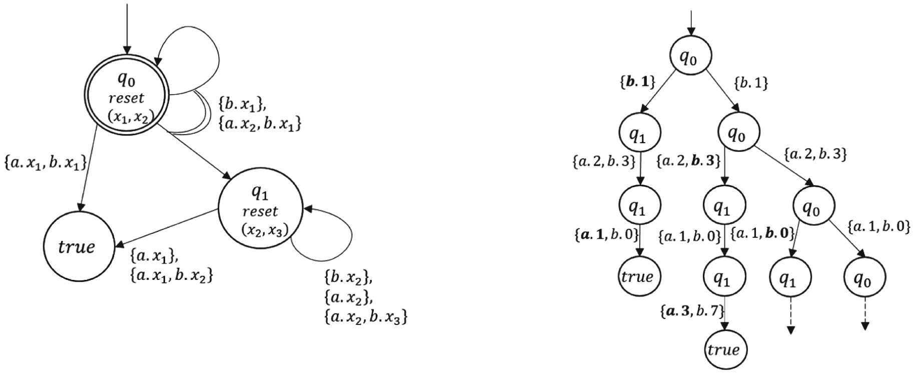

Fig. 1. The AVBW $\mathcal{A}$ described in Example 4 and an example of a run. The double arch between transitions represents an "and" in δ.

- $Q = sub(\varphi')$
- $\forall q \in Q : \{x_p | p \in AP\} \subseteq reset(q)$ and, for $q = \exists x_1, \cdots, \exists x_n \eta$, we have $\{x_1, \cdots, x_n\} \subseteq reset(q)$.
- $E = \{x \neq x' | x' \in X, \exists a : \neg a.x \in sub(\varphi')\}$.
- α consists of all states of the form $\eta \mathsf{V} \psi$.

The set of states Q consists of all sub-formulas of φ'. Intuitively, at every given point there is an assignment to the variables, that may change via resets. If an accepting run of $\mathcal{A}$ on ρ visits a state ψ, then the suffix of ρ that is read from ψ satisfies ψ under the current assignment to the variables. The set of variables X consists of all variables in φ', as well as a variable x_p for every atomic proposition $p \in AP$. The additional variables enable the run to read and ignore currently irrelevant inputs. For example, for $\varphi = \exists x \mathsf{F}\,(b.x \wedge a.x)$, we want to read (and ignore) values of a and b until $a.\gamma \wedge b.\gamma$ occurs with some γ.

Let A be a subset of $AP \times X$ (recall that $\mathcal{B}$ is defined over the alphabet $2^{AP \times X}$). We define δ as follows.

- $\delta(a.x, A) = \mathit{true}$ if $a.x \in A$ and $\delta(a.x, A) = \mathit{false}$ if $a.x \notin A$
- $\delta(\neg a.x, A) = \mathit{true}$ if $a.x \notin A$ and $\delta(\neg a.x, A) = \mathit{false}$ if $a.x \in A$
- $\delta(\eta \wedge \psi, A) = \delta(\eta, A) \wedge \delta(\psi, A)$.
- $\delta(\eta \vee \psi, A) = \delta(\eta, A) \vee \delta(\psi, A)$
- $\delta(\mathsf{X}\,\eta, A) = \eta$
- $\delta(\eta \,\mathsf{U}\, \psi, A) = \delta(\psi, A) \vee (\delta(\eta, A) \wedge \eta \,\mathsf{U}\, \psi)$
- $\delta(\eta \mathsf{V} \psi, A) = \delta(\eta \wedge \psi, A) \vee (\delta(\psi, A) \wedge \eta \mathsf{V} \psi)$
- $\delta(\exists x \eta, A) = \delta(\eta, A)$

Note that since we only use formulas in NNF, we define δ for both *"and"* and *"or"*, as well as for U (until) and V (release) operators.

Correctness. It can be shown that a word ρ is accepted from a state ψ with a variable assignment r iff $\rho \models_r \psi$. We elaborate on how the construction handles the $\exists$-quantifier and negations.

The $\exists$-quantifier is handled by resetting the variables under its scope. Indeed, according to the semantics of $\exists$, for ψ of the form $\exists x : \psi'$, the suffix of ρ holds if ψ' holds for some assignment to x. Resetting x allows the run to correctly assign x in a way that satisfies ψ'. Notice also that from this point on, due to the $\exists$ quantifier, the previous value assigned to x may be forgotten.

Recall that we only allow negations on atomic propositions. We handle these negations with inequalities. If $\neg a.x$ is a sub-formula of φ, then we do not want the value assigned to x to appear with a when reading a from state $\neg a.x$. Thus, all variables that a can occur with from state $\neg a.x$ must be assigned different values from the value currently assigned to x. We express this restriction with the inequality set E . □

3.4 AVBWs Are Not Complementable

As mentioned before, unlike ABWs, AVBWs are not complementable. To prove this, we show that $\forall^*$-VLTL cannot generally be expressed by AVBWs. Since negating an $\exists^*$-VLTL formula produces a $\forall^*$-VLTL formula, the result follows.

Theorem 3. *There is no AVBW that expresses* $\varphi_\forall = \forall x \mathsf{F} a.x$*.*

Proof. Obviously, if the alphabet is not countable, then it cannot be enumerated by a computation. However, the claim holds also for countable alphabets. Assume by way of contradiction that there exists an AVBW $\mathcal{A}$ that expresses $\varphi_\forall$ for $\Gamma = \mathbb{N}$. Then $\mathcal{A}$ accepts $w = a.0a.1a.2\cdots$. Since the variables are not sensitive to their precise contents but only to inequalities among the values, it is easy to see that the accepting run of $\mathcal{A}$ on w can also be used to read $w^1 = a.1a.2\cdots$, in which the value 0 never occurs. □

The negation of the above $\varphi_\forall$ is in $\exists^*$-VLTL, thus there is an AVBW that expresses it.

Corollary 1. *AVBWs are not complementable.*

Corollary 2. $\forall^*$*-VLTL is not expressible by AVBWs.*

3.5 Variable Automata: From AVBW to NVBW

The emptiness problem for NVBWs is NLOGSPACE-complete [10]. In the context of model checking, this is an important property. We now show that for AVBWs, the emptiness problem is undecidable.

Lemma 3. *The emptiness problem for AVBWs is undecidable.*

Proof. According to [17], the satisfiability problem for $\exists^*$-VLTL is undecidable. The satisfiability of a formula φ is equivalent to the nonemptiness of an automaton that expresses φ. Since we have showed that every $\exists^*$-VLTL formula can be expressed by an AVBW, the proof follows. □

Since the emptiness problem for NVBWs is easy, we are motivated to translate AVBWs to NVBWs in order to model check properties that are expressed by AVBWs. In particular, it will enable us to model check $\exists^*$-VLTL properties. This, however, is not possible in general since AVBWs are strictly more expressive than NVBWs (Theorem 1).

In this section we present an incomplete algorithm, which translates an interesting subset of AVBWs to equivalent NVBWs. We later give a structural characterization for AVBWs that can be translated by our algorithm to NVBWs.

3.5.1 From AVBW to NVBW

Our algorithm is inspired by the construction of [14] for translating ABW to NBW. In [14] the states of the NBW are of the form $\langle S, O \rangle$ where S is the set of the states the ABW is currently at, and O is the set of states from paths that "owe" a visit to an accepting state. While running the NBW on a word ρ, accepting states are removed from O, until $O = \emptyset$. Thus, when $O = \emptyset$, all paths have visited an accepting state at least once. Now, O is again set to be S, and a new round begins. The accepting states of the NBW are states of the form $\langle S, \emptyset \rangle$.

Here, we wish to translate an AVBW $\mathcal{A}$ to an NVBW $\mathcal{A}'$. For simplicity, we assume that $E = \emptyset$. The changes for the case where $E \neq \emptyset$ are described later.

In our case, the variables make the translation harder, and as shown before, even impossible in some cases. In addition to S, O we must also remember which variables are currently in use, and might hold values from previous states. In our translation, the states of the NVBW are tuples containing S, O and the sets of variables in use. Since AVBWs allow different paths to assign different values to the same variable, the translation to an NVBW must allocate a new variable for each such assignment. We also need to release variables that were reset in the AVBW, in order to reuse them in the NVBW to avoid defining infinitely many variables. Since we need to know which variables are in use at each step of a run of $\mathcal{A}$, we dynamically allocate both the states and the transitions of $\mathcal{A}'$.

Thus, δ', the transition function of $\mathcal{A}'$, is defined dynamically during the run of our algorithm, as do the states of $\mathcal{A}'$. Moreover, since each path may allocate different values to the same variable, it might be the case that the same variable holds infinitely many values (from different paths). Such a variable induces an unbounded number of variables in $\mathcal{A}'$. Our algorithm halts when no new states are created, and since the fresh variables are part of the created states, creating infinitely many such variables causes our algorithm not to halt. Therefore, the algorithm is incomplete.

Algorithm AVBWtoNVBW: Let $\mathcal{A} = \langle \mathcal{B}_{\mathcal{A}}, \Gamma, E \rangle$ be an AVBW, where $\mathcal{B}_{\mathcal{A}} = \langle 2^X, Q, q_0, \delta, \mathit{reset}, \alpha \rangle$. For simplicity we assume that $\mathcal{B}_{\mathcal{A}}$ is defined over the alphabet 2^X instead of $2^{AP \times X}$. Recall that we assume that $\delta(q, X')$ is in DNF for all $q \in Q, X' \subseteq X$. Let $\mathcal{A}' = \langle \mathcal{B}', \Gamma, E' \rangle$ be an NVBW where $\mathcal{B}' = \langle 2^Z, Q', q_0', \delta', \mathit{reset}', \alpha' \rangle$, and[3]:

- $Z = \{z_i | i = 0..k\}$ is the set of variables. k can be finite or infinite, according to the translation. If $|Z| < \infty$ then the AVBW is translatable to an NVBW.
- $Q' \subseteq 2^{Q \times 2^{X \times Z}} \times 2^{Q \times 2^{X \times Z}}$. The states of $\mathcal{A}'$ are pairs $\langle S, O \rangle$. Each of S, O is a set of pairs of type $\langle q, f_q \rangle$ where $q \in Q$, and $f_q : X \to Z$ is a mapping from the variables of $\mathcal{A}$ to the variables of $\mathcal{A}'$. At each state we need to know how many different values can be assigned to a variable $x \in X$ by $\mathcal{A}$, and create variables $z_i \in Z$ accordingly, in order to keep track of the different values of x.
- $q_0' = \langle \{(q_0, \emptyset)\}, \emptyset \rangle$. The initial state of $\mathcal{A}'$ is the initial state of $\mathcal{A}$ with no additional mappings.
- $\alpha' = 2^{Q \times 2^{X \times Z}} \times \emptyset$. The accepting states of $\mathcal{A}'$ are states for which $O = \emptyset$, i.e., all paths in $\mathcal{A}$ have visited an accepting state.

[3] Comments to the algorithm are given in gray.

1. Preprocessing: For each $q \in Q$: if there is no accepting state or *true* reachable from q then replace q with *false*. This is in order to remove loops that may prevent halting, but, in fact, are redundant since they do not lead to an accepting state.
2. Initiation: set $S = \{\langle q_0, \emptyset \rangle\}, O = \emptyset, Q_{\text{new}} = \{\langle S, O \rangle\}, Q_{\text{old}} = \emptyset, \mathit{vars} = \emptyset, Z = \emptyset$. The purpose of S, O is as explained above; $Q_{\text{new}}, Q_{\text{old}}$ keep track of the changes in the states that the algorithm creates, in order to halt when no new states are created; *vars* holds variables of Z that are currently in use.
3. We iteratively define $\delta'(\langle S, O \rangle, X')$ for $\langle S, O \rangle \in Q_{\text{new}}$, as long as new states are created, i.e. while $Q_{\text{new}} \not\subseteq Q_{\text{old}}$
    - (a) Set: $S' = O' = \emptyset, Z' = \emptyset, Z_{reset} = \emptyset$. The purpose of Z_{reset} is to reset fresh variables, in order to reduce the number of states; at each step, Z' holds the variables in Z which label the current edge (and are the image of the variables in X which label the corresponding edges in $\mathcal{A}$). The group Z_{reset} is initialized at every iteration of the algorithm.
    - (b) $Q_{\text{old}} = Q_{\text{old}} \cup \{\langle S, O \rangle\}$
    - (c) $Q_{\text{new}} = Q_{\text{new}} \setminus \{\langle S, O \rangle\}$
    - (d) For each $\langle q, f_q \rangle \in S$ and $X' \subseteq X$, let $P_q \subseteq Q$ be a minimal set of states with $P_q \vDash \delta(q, X')$.
        - i. Create a state $\langle p, f_p \rangle$ for each $p \in P_q$. The function f_p is initialized to $f_p(x) = f_q(x)$ for $x \notin \mathit{reset}(p)$. I.e., every successor state p of q remembers the same assignments to variables as in q, but releases the assignments to variables that were reset in p.
        - ii. For $x \in X'$ with $x \in \mathit{dom}(f_q)$, update $Z' = Z' \cup \{f_q(x)\}$
        - iii For each $x \in X'$ with $x \notin \mathit{dom}(f_q)$, let $i \in \mathbb{N}$ be the minimal index for which $z_i \notin \mathit{vars}$.
            - A. Add to f_p the mapping $f_p(x) = z_i$ if $x \notin \mathit{reset}(p)$.
            - B. Update $\mathit{vars} = \mathit{vars} \cup \{z_i\}$, $Z' = Z' \cup \{z_i\}$, $Z_{reset} = Z_{reset} \cup \{z_i\}$, $Z = Z \cup \{z_i\}$. z_i may already be in Z, if it was introduced earlier.
        - iv. Define $S_{P_q} = \{\langle p, f_p \rangle\}_{p \in P_q}$
        - v. If $O \neq \emptyset$: define $O_{P_q} = S_{P_q}$ if $\langle q, f_q \rangle \in O$. I.e., add to O' only successors states of states from O.
        - vi. If $O = \emptyset$: define $O_{P_q} = S_{P_q}$
    - (e) Define $S' = \bigcup_{\langle q, f_q \rangle \in S} S_{P_q}$, $O' = (\bigcup_{\langle q, f_q \rangle \in O} O_{P_q}) \setminus \{\langle p, f_p \rangle\}_{p \in \alpha}$
    - (f) Add $\{z_i | z_i \in Z_{reset}\}$ to the reset function of previous state, $\langle S, O \rangle$. I.e., $\mathit{reset}'(\langle S, O \rangle) = \mathit{reset}'(\langle S, O \rangle) \cup \{z_i | z_i \in Z_{reset}\}$.
    - (g) Define $\langle S', O' \rangle \in \delta'(\langle S, O \rangle, Z')$
    - (h) Update $Q_{new} = Q_{new} \cup \{\langle S', O' \rangle\}$
    - (i) If for $z_i \in \mathit{vars}$ it holds that for all $\langle S, O \rangle \in Q_{new}$, for all $\langle p, f_p \rangle \in S$ we have $z_i \notin \mathit{range}(f_q)$, then:
        - i. $\mathit{vars} = \mathit{vars} \setminus \{z_i\}$.
        - ii. add z_i to $\mathit{reset}'(\langle S', O' \rangle)$

      Here we release variables of Z that are no longer in use. The way to do so is to reset them, thus $\mathcal{A}'$ can assign them a new value, and to delete them from *vars* so they can be in use in following transitions.
4. Set $Q' = Q_{old}$

To handle cases where $E \neq \emptyset$, instead of mapping x to any unmapped variable $z_i \in Z$, each variable x may be mapped only to a unique set $\{z_{x_i}\}_{i \in I_x}$. Then, we define $E' = \{z_{x_i} \neq z_{x'_j} | i \in I_x, j \in I_{x'}, (x \neq x') \in E\}$. Notice that this does not change the cardinality of Z.

3.5.2 A Structural Characterization of Translatable AVBWs

In order to define a structural characterization, we wish to refer to an AVBW $\mathcal{A}$ as a directed graph $G_{\mathcal{A}}$ whose nodes are the states of $\mathcal{A}$. There is an edge from q to q' iff q' is in $\delta(q, A)$ for some $A \subseteq X$. For example, if $\delta(q, x) = q_1 \vee (q_2 \wedge q_3)$ then there are edges from q to q_1, q_2 and q_3.

Definition 2. An x-cycle *in an AVBW A is a cycle $q^0, q^1, \cdots, q^k, q^{k+1}$ where $q^{k+1} = q^0$, of states in $G_{\mathcal{A}}$ such that:*

1. *For all $1 \leq i \leq k+1$ it holds that q^i is in $\delta(q^{i-1}, A)$ for some $A \subseteq X$.*
2. *There exists $1 \leq i \leq k+1$ such that q^i is in $\delta(q^{i-1}, A)$ for $A \subseteq X$ and $x \in A$. i.e. there is an edge from one state to another on the cycle, labeled x.*

Theorem 4. *Assume the preprocessing of stage 1 in the algorithm has been applied, resulting in an AVBW $\mathcal{A}$. Algorithm AVBWtoNVBW halts on $\mathcal{A}$ and returns an equivalent NVBW iff for every x-cycle $\mathcal{C}$ in $G_{\mathcal{A}}$ one of the following holds:*

1. *For every q on $\mathcal{C}$ it holds that $x \notin reset(q)$.*
2. *For every state q on a path from the initial state to $\mathcal{C}$ with $q_1 \wedge q_2 \in \delta(q, A)$ for $x \in A$, such that q_1 is on the cycle $\mathcal{C}$ and q_2 leads to an accepting state, it holds that every x-cycle $\mathcal{C}' \neq \mathcal{C}$ on a path from q_2 to an accepting state contains a state q' with $x \in reset(q')$.*

Proof. First, notice that Algorithm AVBWtoNVBW halts iff Z is of a finite size, i.e., the number of variables it produces is finite.

For the first direction we show that running AVBWtoNVBW on an AVBW $\mathcal{A}$ with the above properties results in an NVBW with Z of a finite size. In each of the two cases, we can bound the distance between two reset actions for the same variable, or between a reset action and an accepting state, along every possible run. This, since we can bound the length of the longest path from a state on an x-cycle to an accepting state. Thus all variables in X induce a finite number of variables in Z.

For the other direction, since 1–2 do not hold, there exists a state q that leads both to an x-cycle $\mathcal{C}$ on which x is reset, and to an x-cycle $\mathcal{C}'$ with no $reset(x)$, on a way to an accepting state. While running our algorithm, a new mapping $x \rightarrow z_i$ is introduced at every visit to $reset(x)$ on $\mathcal{C}$. At the same time, z_i cannot be removed from *vars*, because of the visits to $\mathcal{C}'$, which does not reset x, and thus its value must be kept. Therefore, the algorithm continuously creates new assignments $x \rightarrow z_j$ for $j \neq i$. Thus *vars* contains an unbounded set of variables. The fact that there is a path to an accepting state is needed in order for this cycle to "survive" the preprocessing. □

3.5.3 Completeness and Soundness

As we mentioned before, no translation algorithm from AVBWs to NVBWs can be both sound and complete, and have a full characterization of inputs for which the algorithm halts. We now prove this claim.

Theorem 5. *There is no algorithm $\mathcal{E}$ that translates AVBWs into NVBWs such that all the following hold.*

1. *Completeness - for every $\mathcal{A}$ that has an equivalent NVBW, $\mathcal{E}(\mathcal{A})$ halts and returns such an equivalent NVBW.*
2. *Soundness - If $\mathcal{E}(\mathcal{A})$ halts and returns an NVBW $\mathcal{A}'$, then $\mathcal{A}'$ is equivalent to A.*
3. *There is a full characterization of AVBWs for which $\mathcal{E}$ halts.*

Proof. As we have shown in Lemma 3, the emptiness problem of AVBWs is undecidable. Assume there is a translation algorithm $\mathcal{E}$ as described in Theorem 5. Then consider the following procedure. Given an AVBW $\mathcal{A}$, if $\mathcal{E}$ halts, check if $\mathcal{E}(\mathcal{A})$ is empty. If $\mathcal{E}$ does not halt on input $\mathcal{A}$, we know it in advance due to the full characterization. Moreover, we know that $\mathcal{L}(\mathcal{A})$ is not empty (otherwise, since $\mathcal{E}$ is complete, $\mathcal{E}$ would halt on $\mathcal{A}$, since there is an NVBW for the empty language). Hence, a translation algorithm as described in Theorem 5 gives us a procedure to decide the emptiness problem for AVBWs, a contradiction. □

For our algorithm, we have shown a full characterization for halting. Now we prove that our algorithm is sound, and demonstrate its incompleteness by an example of an AVBW for the empty language, for which our algorithm does not halt.

Theorem 6. *Algorithm AVBWtoNVBW is sound.*

Proof. First we show that the definition of E' is correct. Indeed, every $(z_{x_i} \neq z_{x'_j}) \in E'$ is derived from $(x \neq x') \in E$, and each z_{x_i} is induced from only one variable, $x \in X$. Therefore, E' preserves exactly the inequalities of E. Now, $reset'$ is defined according to $reset$ such that if z_i is induced from x, and x is reset in a state q then z_i is reset in states that include q. Therefore $reset'$ allows fresh values only when $reset$ does. The correctness of the rest of the construction follows from the correctness of [14] and from the explanations in the body of the algorithm. □

Example 5. Incompleteness of the algorithm Let $\mathcal{A} = \langle \mathcal{B}, \Gamma, \emptyset \rangle$ where $\mathcal{B} = \langle \{a.x, b.x\}, \{q_0, q_1\}, q_0, \delta, reset, \{q_0\} \rangle$ and $reset(q_0) = \{x\}, reset(q_1) = \emptyset$. The definition of δ is: $\delta(q_0, \{a.x\}) = q_0 \wedge q_1, \delta(q_1, \{a.x\}) = q_1, \delta(q_1, \{b.x\}) = true$. The language of $\mathcal{A}$ is empty, since in order to reach an accepting state on the path from q_1, the input must be exactly $\{b.i\}$ for some $i \in \Gamma$, but the cycle of q_0 only allows to read $\{a.j\}$, without any $b.i$. Although there is an NVBW for the empty language, our algorithm does not halt on $\mathcal{A}$: it keeps allocating new variables to x, thus new states are created and the algorithm does not reach a fixed point.

4 Fragments of $\exists^*$-VLTL Expressible by NVBWs

We now present several sub-fragments of $\exists^*$-VLTL with a direct NVBW construction.

We can construct an NVBW for $\exists^*$-VLTL formula in prenex normal form, denoted *$\exists^*_{pnf}$-VLTL*. The construction relies on the fact that variables cannot change values throughout the run. Since every $\exists^*_{pnf}$-VLTL formula is expressible with an NVBW, together with Lemma 1, we have the following corollary.

Corollary 3. *$\exists^*$-VLTL is stronger than $\exists^*_{pnf}$-VLTL.*[4]*.*

Another easy fragment is *$\exists^*$-(X, F)-VLTL*, which is $\exists^*$-VLTL with only the X, F temporal operators, similar to the definitions of [9]. $\exists$ and X, F are interchangeable. Thus, every $\exists^*$-(X, F)-VLTL formula is equivalent to an $\exists^*_{pnf}$-VLTL, which has a direct construction to an NVBW.

A direct construction from VLTL to NVBWs exists also for $\exists^*$-VLTL formulas in which all quantifiers are either at the beginning of the formula, or adjacent to a parameterized atomic proposition. This extends the construction for $\exists^*_{pnf}$-VLTL by adding resets to some of the states.

5 Model Checking in Practice

The model-checking problem over infinite data domains asks whether an NVBW $\mathcal{A}_M$ accepts a computation that satisfies an $\exists^*$-VLTL formula φ, which specifies "bad" behaviors. If φ is one of the types mentioned in Sect. 4, we can build an equivalent NVBW $\mathcal{A}_\varphi$ for φ. For a general φ, we build an equivalent AVBW $\mathcal{A}$ according to Sect. 3.3 and if the structure of $\mathcal{A}$ agrees with the structural conditions of Theorem 4, we translate $\mathcal{A}$ to an equivalent NVBW $\mathcal{A}_\varphi$ according to Sect. 3.5.1. Now, if $\mathcal{A}_\varphi$ exists, the intersection $\mathcal{A}_\varphi \cap \mathcal{A}_M$ includes all computations of $\mathcal{A}_M$ that are also computations of $\mathcal{A}_\varphi$. Checking the emptiness of the intersection decides whether $\mathcal{A}_M$ has a "bad" behavior that satisfies φ.

6 Conclusions and Future Work

We defined AVBWs, a new model of automata over infinite alphabets that describes all $\exists^*$-VLTL formulas. We showed that AVBWs, unlike ABWs, are not complementable and are stronger than NVBWs. Nevertheless, we presented an algorithm for translating AVBWs to NVBWs when possible, in order to preform model checking. Moreover, we defined a structural characterization of translatable AVBWs. Finally, we presented the full process of model checking a model M given as an NVBW against an $\exists^*$-VLTL formula. A natural extension of our work is to use the techniques presented in this paper in order to preform model checking for VCTL [12] formulas as well.

[4] In [17] the authors conjecture without proof that the formula $\mathsf{G}\,\exists x : a.x$ does not have an equivalent in *PNF*. In Lemma 1 we showed $\mathsf{G}\,\exists x(b.x \wedge \mathsf{F}\,a.x)$ does not have an equivalent NVBW, thus it does not have an equivalent $\exists^*_{pnf}$-VLTL formula. This is a different formula from $\mathsf{G}\,\exists x : a.x$, but the conclusion remains the same.

References

1. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.: Quantified event automata: towards expressive and efficient runtime monitors. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 68–84. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32759-9_9
2. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 14:1–14:64 (2011)
3. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: 21st IEEE Symposium on Logic in Computer Science (LICS 2006), 12–15, Seattle, WA, USA, Proceedings, pp. 7–16. IEEE Computer Society, 2006, August 2006
4. Bouajjani, A., Habermehl, P., Jurski, Y., Sighireanu, M.: Rewriting systems with data. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 1–22. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74240-1_1
5. Brambilla, M., Ceri, S., Comai, S., Fraternali, P., Manolescu, I.: Specification and design of workflow-driven hypertexts. J. Web Eng. **1**(2), 163–182 (2003)
6. J. R. Buechi. On a decision method in restricted second-order arithmetic. In International Congress on Logic, Methodology, and Philosophy of Science, pp. 1–11. Stanford University Press, (1962)
7. Ceri, S., Matera, M., Rizzo, F., Demaldé, V.: Designing data-intensive web applications for content accessibility using web marts. Commun. ACM **50**(4), 55–61 (2007)
8. Colin, S., Mariani, L.: Run-time verification. In: Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.) Model-Based Testing of Reactive Systems. LNCS, vol. 3472, pp. 525–555. Springer, Heidelberg (2005). doi:10.1007/11498490_24
9. Emerson, E.A., Halpern, J.Y.: "sometimes" and "not never" revisited: on branching versus linear time temporal logic. J. ACM **33**(1), 151–178 (1986)
10. Grumberg, O., Kupferman, O., Sheinvald, S.: Variable Automata over Infinite Alphabets. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 561–572. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13089-2_47
11. Grumberg, O., Kupferman, O., Sheinvald, S.: Model checking systems and specifications with parameterized atomic propositions. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, pp. 122–136. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33386-6_11
12. Grumberg, O., Kupferman, O., Sheinvald, S.: A game-theoretic approach to simulation of data-parameterized systems. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 348–363. Springer, Cham (2014). doi:10.1007/978-3-319-11936-6_25
13. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**(2), 329–363 (1994)
14. Miyano, S., Hayashi, T.: Alternating finite automata on omega-words. Theor. Comput. Sci. **32**, 321–330 (1984)
15. Muller, D., Schupp, P.E.: Alternating automata on infinite objects, determinacy and Rabin's theorem. In: Nivat, M., Perrin, D. (eds.) LITP 1984. LNCS, vol. 192, pp. 99–107. Springer, Heidelberg (1985). doi:10.1007/3-540-15641-0_27
16. Neven, F., Schwentick, T., Vianu, V.: Towards regular languages over infinite alphabets. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 560–572. Springer, Heidelberg (2001). doi:10.1007/3-540-44683-4_49

17. Song, F., Wu, Z.: Extending temporal logics with data variable quantifications. In: Raman, V., Suresh, S.P. (eds.) 34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 15–17, 2014, New Delhi, India, vol. 29 of LIPIcs, pp. 253–265. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014, December 2014
18. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996). doi:10.1007/3-540-60915-6_6
19. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: Proceedings of the Symposium on Logic in Computer Science (LICS 1986), Cambridge, Massachusetts, USA, June 16–18, pp. 332–344. IEEE Computer Society (1986)