Alternating Automata Modulo First Order Theories

Radu Iosif, Xiao Xu

Alternating Automata Modulo First Order Theories

Radu Iosif and Xiao Xu

CNRS, Verimag, Université de Grenoble Alpes
Email: {Radu.Iosif,Xiao.Xu}@univ-grenoble-alpes.fr

Abstract. We introduce first-order alternating automata, a generalization of boolean alternating automata, in which transition rules are described by multisorted first-order formulae, with states and internal variables given by uninterpreted predicate terms. The model is closed under union, intersection and complement, and its emptiness problem is undecidable, even for the simplest data theory of equality. To cope with the undecidability problem, we develop an abstraction refinement semi-algorithm based on lazy annotation of the symbolic execution paths with interpolants, obtained by applying (i) quantifier elimination with witness term generation and (ii) Lyndon interpolation in the quantifier-free theory of the data domain, with uninterpreted predicate symbols. This provides a method for checking inclusion of timed and finite-memory register automata, and emptiness of quantified predicate automata, previously used in the verification of parameterized concurrent programs, composed of replicated threads, with shared memory.

1 Introduction

Many results in automata theory rely on the finite alphabet hypothesis, which guarantees, in some cases, the existence of determinization, complementation and inclusion checking methods. However, this hypothesis prevents the use of automata as models of real-time systems or even simple programs, whose input and output are data values ranging over very large domains, typically viewed as infinite mathematical abstractions.

Traditional attempts to generalize classical Rabin-Scott automata to infinite alphabets, such as timed automata [1] and finite-memory automata [16] face the *complement closure* problem: there exist automata for which the complement language cannot be recognized by an automaton in the same class. This makes it impossible to encode a language inclusion problem $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ as the emptiness of an automaton recognizing the language $\mathcal{L}(A) \cap \mathcal{L}^c(B)$, where $\mathcal{L}^c(B)$ denotes the complement of $\mathcal{L}(B)$.

Even for finite alphabets, complementation of finite-state automata faces inherent exponential blowup, due to nondeterminism. However, if we allow universal nondeterminism, in addition to the classical existential nondeterminism, complementation is possible is linear time. Having both existential and universal nondeterminism defines the *alternating automata* model [4]. A finite-alphabet alternating automaton is described by a set of transition rules $q \xrightarrow{a} \phi$, where q is a state, a is an input symbol and ϕ is a boolean formula, whose propositional variables denote successor states.

Our Contribution We extend alternating automata to infinite data alphabets, by defining a model of computation in which all boolean operations, including complementation, can be done in linear time. The control states are given by k-ary predicate symbols

$q(y_1,\ldots,y_k)$, the input consists of an event a from a finite alphabet and a tuple of data variables $x_1,\ldots,x_n$, ranging over an infinite domain, and transitions are of the form $q(y_1,\ldots,y_k) \xrightarrow{a(x_1,\ldots,x_n)} \phi(x_1,\ldots,x_n,y_1,\ldots,y_k)$, where ϕ is a formula in the first-order theory of the data domain. In this model, the arguments of a predicate atom $q(y_1,\ldots,y_k)$ represent the values of the *internal variables* associated with the state. Together with the input values $x_1,\ldots,x_n$, these values define the next configurations, but remain invisible in the input sequence.

The tight coupling of internal values and control states, by means of uninterpreted predicate symbols, allows for linear-time complementation just as in the case of classical propositional alternating automata. Complementation is, moreover, possible when the transition formulae contain first-order quantifiers, generating infinitely-branching execution trees. The price to be paid for this expressivity is that emptiness of first-order alternating automata is undecidable, even for the simplest data theory of equality [6].

The main contribution of this paper is an effective emptiness checking semi-algorithm for first-order alternating automata, in the spirit of the IMPACT lazy annotation procedure, originally developed for checking safety of nondeterministic integer programs [20,21]. In a nutshell, a lazy annotation procedure unfolds an automaton A trying to find an execution that recognizes a word from $\mathcal{L}(A)$. If a path that reaches a final state does not correspond to a concrete run of the automaton, the positions on the path are labeled with interpolants from the proof of infeasibility, thus marking this path and all continuations as infeasible for future searches. Termination of lazy annotation procedures is not guaranteed, but having a suitable coverage relation between the nodes of the search tree may ensure convergence of many real-life examples. However, applying lazy annotation to first-order alternating automata faces two nontrivial problems:

1. Quantified transition rules make it hard, if not impossible, in general, to decide if a path is infeasible. This is mainly because adding uninterpreted predicate symbols to decidable first-order theories, such as Presburger arithmetic, results in undecidability [10]. To deal with this problem, we assume that the first-order data theory, without uninterpreted predicate symbols, has a quantifier elimination procedure, that instantiates quantifers with effectively computable *witness terms*.
2. The interpolants that prove the infeasibility of a path are not *local*, as they may refer to input values encountered in the past. However, the future executions are oblivious to *when* these values have been seen in the past and depend only on the relation between the past and current values. We use this fact to define a labeling of nodes, visited by the lazy annotation procedure, with conjunctions of existentially quantified interpolants combining predicate atoms with data constraints.

We use first-order alternating automata to develop practical semi-algorithms for a number of known undecidable problems, such as: inclusion of regular timed languages [1], inclusion of quasi-regular languages recognized by finite-memory automata [16] and emptiness of predicate automata, a subclass of first-order alternating automata used to verify parameterized concurrent programs [6,7].

Related Work Recognizers for languages over infinite alphabets have found various applications, ranging from Unicode text recognition [5] to runtime program monitoring [2]. Extending finite automata to infinite alphabets has been considered in the context of *symbolic alternating finite automata* (s-AFA), whose transitions are labeled with guards taken from a decidable theory of the data domain [5]. As in our model, s-AFA are closed under union, intersection and complement and emptiness is decidable, due to the lack of registers. However, s-AFA are strictly less expressive than our model, because comparing data at different positions in the input word is not possible.

Constrained Horn clauses (CHC) are a branching computation model widespread in program verification [9]. The main difference between alternating and bottom-up branching computations is that, in an alternating model, all branches of the computation must synchronize on the same input word. With this in mind, it is possible to express emptiness of first-order alternating automata as the existence of solutions of a CHC over a higher-order theory of data, extended with algebraic data types (lists). The effectiveness of such an encoding depends on the effectiveness of interpolation and witness term generation for theories of algebraic data types [11].

The alternating automata model presented in this paper extends the alternating automata with variables ranging over infinite data considered in [14]. There all variables were required to be observable in the input. We overcome this restriction by allowing internal (invisible) variables. Another closely related work [13] considers an inclusion between an asynchronous product of automata $A_1 \times \ldots \times A_n$, extended with data variables, and a monitor automaton B. The semi-algorithm defined there was based on the assumption that all variables of the observer B must be declared in the automata $A_1, \ldots, A_n$ under check. This limitation can now be bypassed, since the inclusion problem can be encoded as emptiness of a first-order alternating automaton and, moreover, the emptiness checking semi-algorithm can handle invisible variables.

The work probably closest to ours concerns the model of *predicate automata* (PA) [6,7,17], used in the verification of parameterized concurrent programs with shared memory. In this model, the alphabet consists of pairs of program statements and thread identifiers and is considered infinite because the number of threads is unbounded. Since thread identifiers can only be compared for equality, the data theory in PA is the theory of equality. Even with this simplification, the emptiness problem is undecidable when either the predicates have arity greater than one [6] or use quantified transition rules [17]. Checking emptiness of quantifier-free PA is possible semi-algorithmically, by explicitly enumerating reachable configurations and checking coverage by looking for permutations of argument values. However, no semi-algorithm has been given for quantified PA. Dealing with quantified transition rules is one of our contributions.

1.1 Preliminaries

For two integers $0 \leq i \leq j$, we define $[i, j] \overset{\mathsf{def}}{=} \{i, \ldots, j\}$ and $[i] \overset{\mathsf{def}}{=} [0, i]$. We consider two disjoint sorts $\mathbb{D}$ and $\mathbb{B}$, where $\mathbb{D}$ is an infinite domain and $\mathbb{B} = \{\top, \bot\}$ is the set of boolean values true ($\top$) and false ($\bot$), respectively. The $\mathbb{D}$ sort is equipped with countably many function symbols $f : \mathbb{D}^{\#(f)} \to \mathbb{D} \cup \mathbb{B}$, where $\#(f) \geq 0$ denotes the number of arguments (arity) of f. A *predicate* is a function symbol $p : \mathbb{D}^{\#(p)} \to \mathbb{B}$ that is, a $\#(p)$-ary relation.

We consider the interpretation of all function symbols $f : \mathbb{D}^{\#(f)} \to \mathbb{D}$ to be fixed by the interpretation of the $\mathbb{D}$ sort, for instance if $\mathbb{D}$ is the set of integers $\mathbb{Z}$, these are zero, the successor function and the arithmetic operations of addition and multiplication. We extend this convention to several predicates over $\mathbb{D}$, such as the inequality relation over $\mathbb{Z}$, and write Pred for the set of remaining *uninterpreted predicates*.

Let $\mathsf{Var} = \{x, y, z, \ldots\}$ be a countably infinite set of variables, ranging over $\mathbb{D}$. Terms are either constants of sort $\mathbb{D}$, variables or function applications $f(t_1, \ldots, t_{\#(f)})$, where $t_1, \ldots, t_{\#(f)}$ are terms. The set of first-order formulae is defined by the syntax below:

$$\phi := t = s \mid p(t_1, \ldots, t_{\#(p)}) \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \exists x \,.\, \phi_1$$

where $t, s, t_1, \ldots, t_{\#(p)}$ denote terms and p is a predicate symbol. We write $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$ and $\forall x \,.\, \phi_1$ for $\neg(\neg\phi_1 \wedge \neg\phi_2)$, $\neg\phi_1 \vee \phi_2$ and $\neg\exists x \,.\, \neg\phi_1$, respectively. $\mathsf{FV}(\phi)$ is the set of free variables in ϕ and the size $|\phi|$ of a formula ϕ is the number of symbols needed to write it down. A *sentence* is a formula ϕ with no free variables. A formula is *positive* if each uninterpreted predicate symbol occurs under an even number of negations and we denote by $\mathsf{Form}^+(Q, X)$ the set of positive formulae with predicates from the set $Q \subseteq \mathsf{Pred}$ and free variables from the set $X \subseteq \mathsf{Var}$. A formula is in *prenex form* if it is of the form $\varphi = Q_1 x_1 \ldots Q_n x_n \,.\, \phi$, where ϕ has no quantifiers. In this case we call ϕ the *matrix* of φ. Every first-order formula can be written in prenex form, by renaming each quantified variable to a unique name and moving the quantifiers upfront.

An *interpretation* $\mathcal{I}$ maps each predicate symbol p into a set $p^{\mathcal{I}} \subseteq \mathbb{D}^{\#(p)}$, if $\#(p) > 0$, or into an element of $\mathbb{B}$ if $\#(p) = 0$. A *valuation* ν maps each variable x into an element of $\mathbb{D}$. Given a term t, we denote by t^{ν} the value obtained by replacing each variable x by the value $\nu(x)$ and evaluating each function application. For a formula ϕ, we define the forcing relation $\mathcal{I}, \nu \models \phi$ recursively on the structure of ϕ, as usual. For a formula ϕ and a valuation ν, we define $[\![\phi]\!]_{\nu} \overset{\mathsf{def}}{=} \{\mathcal{I} \mid \mathcal{I}, \nu \models \phi\}$ and drop the ν subscript for sentences. A sentence ϕ is *satisfiable* if $[\![\phi]\!] \neq \emptyset$. An element of $[\![\phi]\!]$ is called a *model* of ϕ. A formula ϕ is *valid* if $\mathcal{I}, \nu \models \phi$ for every interpretation $\mathcal{I}$ and every valuation ν. We say that ϕ *entails* ψ, written $\phi \models \psi$ if and only if $[\![\phi]\!] \subseteq [\![\psi]\!]$.

Interpretations are partially ordered by the pointwise subset order, defined as $\mathcal{I}_1 \subseteq \mathcal{I}_2$ if and only if $p^{\mathcal{I}_1} \subseteq p^{\mathcal{I}_2}$ for each predicate symbol $p \in \mathsf{Pred}$. Given a formula ϕ and a valuation ν, we define $[\![\phi]\!]^{\mu}_{\nu} \overset{\mathsf{def}}{=} \{\mathcal{I} \mid \mathcal{I}, \nu \models \phi,\ \forall \mathcal{I}' \subseteq \mathcal{I} \,.\, \mathcal{I}', \nu \not\models \phi\}$ the set of minimal interpretations that, together with ν, form models of ϕ.

2 First Order Alternating Automata

Let Σ be a finite alphabet Σ of *input events*. Given a finite set of variables $X \subseteq \mathsf{Var}$, we denote by $X \mapsto \mathbb{D}$ the set of valuations of the variables X and $\Sigma[X] = \Sigma \times (X \mapsto \mathbb{D})$ be the possibly infinite set of *data symbols* (a, ν), where a is an input symbol and ν is a valuation. A *data word* (simply called word in the following) is a finite sequence $w = (a_1, \nu_1)(a_2, \nu_2) \ldots (a_n, \nu_n)$ of data symbols. Given a word w, we denote by $w_{\Sigma} \overset{\mathsf{def}}{=} a_1 \ldots a_n$ its sequence of input events and by $w_{\mathbb{D}}$ the valuation associating each time-stamped variable $x^{(i)}$, where $x \in \mathsf{Var}$, the value $\nu_i(x)$, for all $i \in [1, n]$. We denote by ε the empty sequence, by Σ^* the set of finite input sequences and by $\Sigma[X]^*$ the set of finite data words over the variables X.

A *first-order alternating automaton* is a tuple $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$, where Σ is a finite set of input events, X is a finite set of input variables, Q is a finite set of predicates denoting control states, $\iota \in \mathsf{Form}^+(Q, \emptyset)$ is a sentence defining initial configurations, $F \subseteq Q$ is the set of predicates denoting final states and Δ is a set of *transition rules*. A transition rule is of the form $q(y_1, \ldots, y_{\#(q)}) \xrightarrow{a(X)} \psi$, where $q \in Q$ is a predicate, $a \in \Sigma$

is an input event and $\psi \in \mathsf{Form}^+(Q, X \cup \{y_1,\ldots,y_{\#(q)}\})$ is a positive formula, where $X \cap \{y_1,\ldots,y_{\#(q)}\} = \emptyset$. Without loss of generality, we consider, for each predicate $q \in Q$ and each input event $a \in \Sigma$, at most one such rule, as two or more rules can be joined using disjunction. The quantifiers occurring in the right-hand side formula of a transition rule are called *transition quantifiers*. The *size* of $\mathcal{A}$ is $|\mathcal{A}| \stackrel{\mathsf{def}}{=} |\iota| + \sum\{|\psi| \mid q(\mathbf{y}) \xrightarrow{a(X)} \psi \in \Delta\}$.

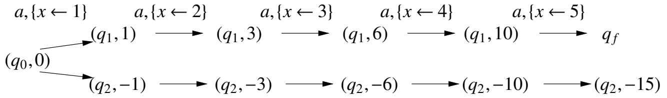
The semantics of first-order alternating automata is analogous to the semantics of propositional alternating automata, with rules of the form $q \xrightarrow{a} \phi$, where q is a propositional variable and ϕ a positive boolean combination of propositional variables. For instance, $q_0 \xrightarrow{a} (q_1 \wedge q_2) \vee q_3$ means that the automaton can choose to transition in either both q_1 and q_2 or in q_3 alone. This leads to defining transitions as the *minimal models* of the right hand side of a rule[1]. The original definition of alternating automata [4] works around this problem and considers boolean valuations instead of formulae. In contrast, a finite description of a first-order alternating automaton cannot be given in terms of interpretations, as a first-order formula may have infinitely many models, corresponding to infinitely many initial or successor states occurring within an execution step.

Given an uninterpreted predicate symbol $q \in Q$ and data values $d_1,\ldots,d_{\#(q)} \in \mathbb{D}$, the tuple $(q, d_1,\ldots,d_{\#(q)})$ is called a *configuration*, sometimes written $q(d_1,\ldots,d_{\#(q)})$, when no confusion arises. A configuration is *final* if $q \in F$. An interpretation $\mathcal{I}$ corresponds to a set of configurations $\mathsf{c}(\mathcal{I}) \stackrel{\mathsf{def}}{=} \{(q, d_1,\ldots,d_{\#(q)}) \mid q \in Q,\ (d_1,\ldots,d_{\#(q)}) \in q^{\mathcal{I}}\}$, called a *cube*. This notation is lifted to sets of configurations in the usual way.

Definition 1. *Given a word $w = (a_1,\nu_1)\ldots(a_n,\nu_n) \in \Sigma[X]^*$ and a cube c, an* execution *of $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$ over w, starting with c, is a forest $\mathcal{T} = \{T_1, T_2, \ldots\}$, where each T_i is a tree labeled with configurations, such that:*

1. *$c = \{T(\epsilon) \mid T \in \mathcal{T}\}$ is the set of configurations labeling the roots of $T_1, T_2, \ldots$ and*
2. *if $(q, d_1,\ldots,d_{\#(q)})$ labels a node on the level $j \in [n-1]$ in T_i, then the labels of its children form a cube from $\mathsf{c}([\![\psi]\!]^{\mu}_{\eta})$, where $\eta = \nu_{j+1}[y_1 \leftarrow d_1,\ldots,y_{\#(q)} \leftarrow d_{\#(q)}]$ and $q(y_1,\ldots,y_{\#(q)}) \xrightarrow{a_{j+1}(X)} \psi \in \Delta$ is a transition rule of $\mathcal{A}$.*

An execution $\mathcal{T}$ over w, starting with c, is *accepting* if and only if all paths in $\mathcal{T}$ have the same length and the frontier of each tree $T \in \mathcal{T}$ is labeled with final configurations. If $\mathcal{A}$ has an accepting execution over w starting with a cube $c \in \mathsf{c}([\![\iota]\!]^{\mu})$, then $\mathcal{A}$ *accepts* w and let $\mathcal{L}(\mathcal{A})$ be the set of words accepted by $\mathcal{A}$. For example, consider the automaton $\mathcal{A} = \langle \{a\}, \{x\}, \{q_0, q_1, q_2, q_f\}, q_0(0), \{q_f\}, \Delta \rangle$, where Δ is the set: $q_0(y) \xrightarrow{a(x)} q_1(y+x) \wedge q_2(y-x)$, $q_1(y) \xrightarrow{a(x)} q_1(y+x) \vee (y > 0 \wedge q_f)$ and $q_2(y) \xrightarrow{a(x)} q_2(y-x) \vee (y > 0 \wedge q_f)$. A possible execution tree of this automaton is the following:



The execution tree is not accepting, since its frontier is not labeled with final configurations everywhere. Incidentally, here we have $\mathcal{L}(\mathcal{A}) = \emptyset$, which is proved by our tool in ~ 0.5 seconds on an average machine.

[1] Both $\{q_1 \leftarrow \top, q_2 \leftarrow \top, q_3 \leftarrow \bot\}$ and $\{q_1 \leftarrow \bot, q_2 \leftarrow \bot, q_3 \leftarrow \top\}$ are minimal models, however $\{q_1 \leftarrow \top, q_2 \leftarrow \top, q_3 \leftarrow \top\}$ is a model but is not minimal.

In the rest of this paper, we are concerned with the following problems:

1. *boolean closure*: given automata $\mathcal{A}_i = \langle \Sigma, X, Q_i, \iota_i, F_i, \Delta_i \rangle$, for $i = 1, 2$, do there exist automata $\mathcal{A}_\cap$, $\mathcal{A}_\cup$ and $\overline{\mathcal{A}}_1$ such that $L(\mathcal{A}_\cap) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, $L(\mathcal{A}_\cup) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ and $L(\overline{\mathcal{A}}_1) = \Sigma[X]^* \setminus L(\mathcal{A}_1)$?
2. *emptiness*: given an automaton $\mathcal{A}$, is $L(\mathcal{A}) = \emptyset$?

For technical reasons, we address the following problem next: given an automaton $\mathcal{A}$ and an input sequence $\alpha \in \Sigma^*$, does there exists a word $w \in \mathcal{L}(\mathcal{A})$ such that $w_\Sigma = \alpha$? By solving this problem first, we develop the machinery required to prove that first-order alternating automata are closed under complement and, further, set up the ground for developping a practical semi-algorithm for the emptiness problem.

2.1 Path Formulae

In the upcoming developments it is sometimes more convenient to work with logical formulae defining executions of automata, than with low-level execution forests. For this reason, we first introduce *path formulae* $\Theta(\alpha)$, which are formulae defining the executions of an automaton, over words that share a given sequence α of input events. Second, we restrict a path formula $\Theta(\alpha)$ to an *acceptance formula* $\Upsilon(\alpha)$, which defines only those executions that are accepting among $\Theta(\alpha)$. Consequently, the automaton accepts a word w such that $w_\Sigma = \alpha$ if and only if $\Upsilon(\alpha)$ is satisfiable.

Let $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$ be an automaton for the rest of this section. For any $i \in \mathbb{N}$, we denote by $Q^{(i)} = \{q^{(i)} \mid q \in Q\}$ and $X^{(i)} = \{x^{(i)} \mid x \in X\}$ the sets of time-stamped predicate symbols and variables, respectively. We also define $Q^{(\leq n)} \stackrel{\mathsf{def}}{=} \{q^{(i)} \mid q \in Q, i \in [n]\}$ and $X^{(\leq n)} \stackrel{\mathsf{def}}{=} \{x^{(i)} \mid x \in X, i \in [n]\}$. For a formula ψ and $i \in \mathbb{N}$, we define $\psi^{(i)} \stackrel{\mathsf{def}}{=} \psi[X^{(i)}/X, Q^{(i)}/Q]$ the formula in which all input variables and state predicates (and only those symbols) are replaced by their time-stamped counterparts. Moreover, we write $q(\mathbf{y})$ for $q(y_1, \ldots, y_{\#(q)})$, when no confusion arises.

Given a sequence of input events $\alpha = a_1 \ldots a_n \in \Sigma^*$, the *path formula* of α is:

$$\Theta(\alpha) \stackrel{\mathsf{def}}{=} \iota^{(0)} \wedge \bigwedge_{i=1}^{n} \bigwedge_{q(\mathbf{y}) \xrightarrow{a_i(X)} \psi \in \Delta} \forall y_1 \ldots \forall y_{\#(q)} \,.\, q^{(i-1)}(\mathbf{y}) \rightarrow \psi^{(i)} \tag{1}$$

The automaton $\mathcal{A}$, to which $\Theta(\alpha)$ refers, will always be clear from the context. To formalize the relation between the low-level configuration-based execution semantics and path formulae, consider a word $w = (a_1, \nu_1) \ldots (a_n, \nu_n) \in \Sigma[X]^*$. Any execution $\mathcal{T}$ of $\mathcal{A}$ over w has an associated interpretation $\mathcal{I}_\mathcal{T}$ of time-stamped predicates $Q^{(\leq n)}$:

$$\mathcal{I}_\mathcal{T}(q^{(i)}) \stackrel{\mathsf{def}}{=} \{(d_1, \ldots, d_{\#(q)}) \mid (q, d_1, \ldots, d_{\#(q)}) \text{ labels a node on level } i \text{ in } \mathcal{T}\},\ \forall q \in Q\ \forall i \in [n]$$

Lemma 1. *Given an automaton $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$, for any word $w = (a_1, \nu_1) \ldots (a_n, \nu_n)$, we have $[\![\Theta(w_\Sigma)]\!]^{\mu}_{w_\mathbb{D}} = \{\mathcal{I}_\mathcal{T} \mid \mathcal{T}$ is an execution of $\mathcal{A}$ over $w\}$.*

Next, we give a logical characterization of acceptance, relative to a given sequence of input events $\alpha \in \Sigma^*$. To this end, we constrain the path formula $\Theta(\alpha)$ by requiring that only final states of $\mathcal{A}$ occur on the last level of the execution. The result is the *acceptance formula* for α:

$$\Upsilon(\alpha) \stackrel{\mathsf{def}}{=} \Theta(\alpha) \wedge \bigwedge_{q \in Q \setminus F} \forall y_1 \ldots \forall y_{\#(q)} \,.\, q^{(n)}(\mathbf{y}) \rightarrow \bot \tag{2}$$

The top-level universal quantifiers from a subformula $\forall y_1 \ldots \forall y_{\#(q)} \,.\, q^{(i)}(\mathbf{y}) \rightarrow \psi$ of $\Upsilon(\alpha)$ will be referred to as *path quantifiers*, in the following. Notice that path quantifiers are distinct from the transition quantifiers that occur within a formula ψ of a transition rule $q(y_1,\dots,y_{\#(q)}) \xrightarrow{a(X)} \psi$ of $\mathcal{A}$. The relation between the words accepted by $\mathcal{A}$ and the acceptance formula above, is formally captured by the following lemma:

Lemma 2. *Given an automaton $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$, for every word $w \in \Sigma[X]^*$, the following are equivalent: (1) there exists an interpretation $\mathcal{I}$ such that $\mathcal{I}, w_{\mathbb{D}} \models \Upsilon(w_\Sigma)$ and (2) $w \in \mathcal{L}(\mathcal{A})$.*

As an immediate consequence, one can decide whether $\mathcal{A}$ accepts some word w with a given input sequence $w_\Sigma = \alpha$, by checking whether $\Upsilon(\alpha)$ is satisfiable. However, unlike non-alternating infinite-state models of computation, such as counter automata (nondeterministic programs with integer variables), the satisfiability query for an acceptance (path) formula falls outside of known decidable theories, supported by standard SMT solvers. There are basically two reasons for this, namely (i) the presence of predicate symbols, and (ii) the non-trivial alternation of quantifiers. To understand this point, consider for example, the decidable theory of Presburger arithmetic [24]. Adding even only one monadic predicate symbol to it yields undecidability in the presence of non-trivial quantifier alternation [10]. On the other hand, the quantifier-free fragment of Presburger arithmetic extended with uninterpreted function symbols is decidable, by a Nelson-Oppen style congruence closure argument [22].

To tackle the problem of deciding satisfiability of $\Upsilon(\alpha)$ formulae, we start from the observation that their form is rather particular, which allows the elimination of path quantifiers and uninterpreted predicate symbols, by a couple of satisfiability-preserving transformations. The result of applying these transformations is a formula with no predicate symbols, whose only quantifiers are those introduced by the transition rules of the automaton. Next, in §3 we shall assume moreover that the first-order theory of the data sort $\mathbb{D}$ (without uninterpreted predicate symbols) has quantifier elimination, providing thus an effective decision procedure.

For the time being, let us formally define the elimination of transition quantifiers and predicate symbols. Let $\alpha = a_1 \dots a_n$ be a given sequence of input events and let α_i be the prefix $a_1 \dots a_i$ of α, for $i \in [n]$, where $\alpha_0 = \epsilon$. We consider the sequence of formulae $\widehat{\Theta}(\alpha_0), \dots, \widehat{\Theta}(\alpha_n)$ defined as $\widehat{\Theta}(\alpha_0) \overset{\mathsf{def}}{=} \iota^{(0)}$ and, for all $i \in [1,n]$, let $\widehat{\Theta}(\alpha_i)$ be the conjunction of $\widehat{\Theta}(\alpha_{i-1})$ with all formulae $q^{(i-1)}(t_1,\dots,t_{\#(q)}) \rightarrow \psi^{(i)}[t_1/y_1,\dots,t_{\#(q)}/y_{\#(q)}]$, such that $q^{(i-1)}(t_1,\dots,t_{\#(q)})$ occurs in $\widehat{\Theta}(\alpha_{i-1})$, for some terms $t_1,\dots,t_{\#(q)}$. Next, we write $\widehat{\Upsilon}(\alpha)$ for the conjunction of $\widehat{\Theta}(\alpha_n)$ with all $q^{(n)}(t_1,\dots,t_{\#(q)}) \rightarrow \bot$, such that $q^{(n)}(t_1,\dots,t_{\#(q)})$ occurs in $\widehat{\Theta}(\alpha_n)$, for some $q \in Q \setminus F$. Note that $\widehat{\Upsilon}(\alpha)$ contains no path quantifiers, as required. On the other hand, the scope of the transition quantifiers in $\widehat{\Upsilon}(\alpha)$ exceeds the right-hand side formulae from the transition rules, as shown by the following example.

Example 1. Consider the automaton $\mathcal{A} = \langle \{a_1,a_2\}, \{x\}, \{q,q_f\}, \iota, \{q_f\}, \Delta \rangle$, where:

$$\iota = \exists z \,.\, z \geq 0 \wedge q(z)$$

$$\Delta = \{q(y) \xrightarrow{a_1(x)} x \geq 0 \wedge \forall z \,.\, z \leq y \rightarrow q(x+z),\ q(y) \xrightarrow{a_2(x)} y < 0 \wedge q_f(x+y)\}$$

For the input event sequence $\alpha = a_1 a_2$, the acceptance formula is:

$$\begin{aligned}\Upsilon(\alpha) = {} & \exists z_1 \,.\, z_1 \geq 0 \wedge q^{(0)}(z_1) \wedge \\ & \forall y \,.\, q^{(0)}(y) \rightarrow [x^{(1)} \geq 0 \wedge \forall z_2 \,.\, z_2 \geq y \rightarrow q^{(1)}(x^{(1)} + z_2)] \wedge \\ & \forall y \,.\, q^{(1)}(y) \rightarrow [y < 0 \wedge {q_f}^{(2)}(x^{(2)} + y)]\end{aligned}$$

The result of eliminating the path quantifiers, in prenex normal form, is shown below:

$$\begin{aligned}\widehat{\varUpsilon}(\alpha) = \exists z_1 \forall z_2 \,.\, & z_1 \geq 0 \wedge q^{(0)}(z_1) \wedge \\ & [q^{(0)}(z_1) \rightarrow x^{(1)} \geq 0 \wedge (z_2 \geq z_1 \rightarrow q^{(1)}(x^{(1)} + z_2))] \wedge \\ & [q^{(1)}(x^{(1)} + z_2) \rightarrow x^{(1)} + z_2 < 0 \wedge {q_f}^{(2)}(x^{(2)} + x^{(1)} + z_2)]\end{aligned}$$

Notice that the transition quantifiers $\exists z_1$ and $\forall z_2$ from $\varUpsilon(\alpha)$ range now over $\widehat{\varUpsilon}(\alpha)$. ■

Lemma 3. *For any input event sequence $\alpha = a_1 \ldots a_n$ and each valuation $\nu : X^{(\leq n)} \rightarrow \mathbb{D}$, the following hold, for every interpretation $\mathcal{I}$: (1) if $\mathcal{I}, \nu \models \varUpsilon(\alpha)$ then $\mathcal{I}, \nu \models \widehat{\varUpsilon}(\alpha)$, and (2) if $\mathcal{I}, \nu \models \widehat{\varUpsilon}(\alpha)$ there exists an interpretation $\mathcal{J} \subseteq \mathcal{I}$ such that $\mathcal{J}, \nu \models \varUpsilon(\alpha)$.*

Further, we eliminate the predicate atoms from $\widehat{\varUpsilon}(\alpha)$, by considering the sequence of formulae $\overline{\varTheta}(\alpha_0) \overset{\text{def}}{=} \iota^{(0)}$ and $\overline{\varTheta}(\alpha_i)$ is obtained by substituting each predicate atom $q^{(i-1)}(t_1, \ldots, t_{\#(q)})$ in $\overline{\varTheta}(\alpha_{i-1})$ by $\psi^{(i)}[t_1/y_1, \ldots, t_{\#(q)}/y_{\#(q)}]$, where $q(\mathbf{y}) \xrightarrow{a_i(X)} \psi \in \varDelta$, for all $i \in [1,n]$. We write $\overline{\varUpsilon}(\alpha)$ for the formula obtained by replacing, in $\overline{\varTheta}(\alpha)$, each occurrence of a predicate $q^{(n)}$, such that $q \in Q \setminus F$ (resp. $q \in F$), by $\bot$ (resp. $\top$).

Example 2 (Contd. from Example 1*).* The result of the elimination of predicate atoms from the acceptance formula in Example 1 is shown below:

$$\overline{\varUpsilon}(\alpha) = \exists z_1 \forall z_2 \,.\, z_1 \geq 0 \wedge [x^{(1)} \geq 0 \wedge (z_2 \geq z_1 \rightarrow x^{(1)} + z_2 < 0)]$$

Since this formula is unsatisfiable, by Lemma 5 below, no word w with input event sequence $w_\Sigma = a_1 a_2$ is accepted by the automaton $\mathcal{A}$ from Example 1. ■

At this point, we prove the formal relation between the satisfiability of the formulae $\widehat{\varUpsilon}(\alpha)$ and $\overline{\varUpsilon}(\alpha)$. Since there are no occurrences of predicates in $\overline{\varUpsilon}(\alpha)$, for each valuation $\nu : X^{(\leq n)} \rightarrow \mathbb{D}$, there exists an interpretation $\mathcal{I}$ such that $\mathcal{I}, \nu \models \overline{\varUpsilon}(\alpha)$ if and only if $\mathcal{J}, \nu \models \overline{\varUpsilon}(\alpha)$, for every interpretation $\mathcal{J}$. In this case we omit $\mathcal{I}$ and simply write $\nu \models \overline{\varUpsilon}(\alpha)$.

Lemma 4. *For any input event sequence $\alpha = a_1 \ldots a_n$ and each valuation $\nu : X^{(\leq n)} \rightarrow \mathbb{D}$, there exists a valuation $\mathcal{I}$ such that $\mathcal{I}, \nu \models \widehat{\varUpsilon}(\alpha)$ if and only if $\nu \models \overline{\varUpsilon}(\alpha)$.*

Finally, we define the acceptance of a word with a given input event sequence by means of a quantifier-free formula in which no predicate atom occurs.

Lemma 5. *Given an automaton $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \varDelta \rangle$, for every word $w \in \Sigma[X]^*$, we have $w_{\mathbb{D}} \models \overline{\varUpsilon}(w_\Sigma)$ if and only if $w \in \mathcal{L}(\mathcal{A})$.*

2.2 Boolean Closure of First Order Alternating Automata

Given a positive formula ϕ, we define the *dual* formula $\phi^\sim$ recursively as follows:

$$\begin{array}{lll} (\phi_1 \vee \phi_2)^\sim \overset{\text{def}}{=} {\phi_1}^\sim \wedge {\phi_2}^\sim & (\phi_1 \wedge \phi_2)^\sim \overset{\text{def}}{=} {\phi_1}^\sim \vee {\phi_2}^\sim & (t = s)^\sim \overset{\text{def}}{=} t \neq s \\ (\exists x \,.\, \phi_1)^\sim \overset{\text{def}}{=} \forall x \,.\, {\phi_1}^\sim & (\forall x \,.\, \phi_1)^\sim \overset{\text{def}}{=} \exists x \,.\, {\phi_1}^\sim & (t \neq s)^\sim \overset{\text{def}}{=} t = s \end{array}$$

$$q(x_1, \ldots, x_{\#(q)})^\sim \overset{\text{def}}{=} q(x_1, \ldots, x_{\#(q)})$$

The following theorem shows closure of automata under all boolean operations. Note that it is sufficient to show closure under intersection and negation because $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$ is the complement of the language $\mathcal{L}^c(\mathcal{A}_1) \cap \mathcal{L}^c(\mathcal{A}_2)$, for any two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ with the same input event alphabet and set of input variables.

Theorem 1. *Given automata $\mathcal{A}_i = \langle \Sigma, X, Q_i, \iota_i, F_i, \Delta_i \rangle$, for $i = 1,2$, such that $Q_1 \cap Q_2 = \emptyset$, the following hold:*

1. *$\mathcal{L}(\mathcal{A}_\cap) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$, where $\mathcal{A}_\cap = \langle \Sigma, X, Q_1 \cup Q_2, \iota_1 \wedge \iota_2, F_1 \cup F_2, \Delta_1 \cup \Delta_2 \rangle$,*
2. *$\mathcal{L}(\overline{\mathcal{A}_i}) = \Sigma[X]^* \setminus \mathcal{L}(\mathcal{A}_i)$, where $\overline{\mathcal{A}_i} = \langle \Sigma, X, Q_i, \iota^\sim, Q_i \setminus F_i, \Delta_i^\sim \rangle$ and $\Delta_i^\sim = \{q(\mathbf{y}) \xrightarrow{a(X)} \psi^\sim \mid q(\mathbf{y}) \xrightarrow{a(X)} \psi \in \Delta_i\}$, for $i = 1,2$.*

Moreover, $|\mathcal{A}_\cap| = \mathcal{O}(|\mathcal{A}_1| + |\mathcal{A}_2|)$ and $|\overline{\mathcal{A}_i}| = \mathcal{O}(|\mathcal{A}_i|)$, for $i = 1,2$.

3 The Emptiness Problem

The emptiness problem is undecidable even for automata with predicates of arity two, whose transition rules use only equalities and disequalities, having no transition quantifiers [6]. Since even such simple classes of alternating automata have no general decision procedure for emptiness, we use an abstraction-refinement semi-algorithm based on *lazy annotation* [20,21]. In a nutshell, a lazy annotation procedure systematically explores the set of finite input event sequences searching for an accepting execution. For an input sequence, if the path formula is satisfiable, we compute a word in the language of the automaton, from the model of the path formula. Otherwise, i.e. the sequence is *spurious*, the search backtracks and each position in the sequence is annotated with an interpolant, thus marking the sequence as infeasible. The semi-algorithm uses moreover a coverage relation between sequences, ensuring that the continuations of already covered sequences are never explored. Sometimes this coverage relation provides a sound termination argument, in case when the automaton is empty.

For two input event sequences $\alpha, \beta \in \Sigma^*$, we say that α is a prefix of β, written $\alpha \preceq \beta$, if $\alpha = \beta\gamma$ for some sequence $\gamma \in \Sigma^*$. A set S of sequences is *prefix-closed* if for each $\alpha \in S$, if $\beta \preceq \alpha$ then $\beta \in S$, and *complete* if for each $\alpha \in S$, there exists $a \in \Sigma$ such that $\alpha a \in S$ if and only if $\alpha b \in S$ for all $b \in \Sigma$. A prefix-closed set is the backbone of a tree whose edges are labeled with input events. If the set is, moreover, complete, then every node of the tree has either zero successors, in which case it is called a *leaf*, or it has a successor edge labeled with a for each input event $a \in \Sigma$.

Definition 2. *An* unfolding *of an automaton $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$ is a finite partial mapping $U : \Sigma^* \rightharpoonup_{fin} \mathsf{Form}^+(Q, \emptyset)$, whose domain* $\mathrm{dom}(U)$ *is a finite prefix-closed complete set, such that $U(\epsilon) = \iota$, and for each sequence $\alpha a \in$* $\mathrm{dom}(U)$, *such that $\alpha \in \Sigma^*$ and $a \in \Sigma$:*

$$U(\alpha)^{(0)} \wedge \bigwedge_{q(\mathbf{y}) \xrightarrow{a(X)} \psi} \forall y_1 \ldots \forall y_{\#q} \,.\, q^{(0)}(\mathbf{y}) \rightarrow \psi^{(1)} \models U(\alpha a)^{(1)}$$

A path α is safe *in U if and only if $U(\alpha) \wedge \bigwedge_{q \in Q \setminus F} \forall y_1 \ldots \forall_{y_{\#(q)}} \,.\, q(\mathbf{y}) \rightarrow \bot$ is unsatisfiable. The unfolding U is safe if and only if every path in* $\mathrm{dom}(U)$ *is safe in U.*

Lazy annotation semi-algorithms [20,21] build unfoldings of automata trying to discover counterexamples for emptiness. If the automaton $\mathcal{A}$ in question is non-empty, a systematic enumeration of the input event sequences[2] from Σ^* will suffice to discover a word $w \in \mathcal{L}(\mathcal{A})$, provided that the first-order theory of the data domain $\mathbb{D}$ is decidable (Lemma 2). However, if $\mathcal{L}(\mathcal{A}) = \emptyset$, the enumeration of input event sequences may, in principle, run forever. The typical way of fighting this divergence problem is to define a *coverage* relation between the nodes of the unfolding tree.

[2] For instance, using breadth-first search.

Algorithm 1 IMPACT-based Semi-algorithm for First Order Alternating Automata

input: a first order alternating automaton $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$

output: $\top$ if $L(\mathcal{A}) = \emptyset$, or word $w \in L(\mathcal{A})$, otherwise

data structures: `WorkList` and unfolding tree $\mathcal{U} = \langle N, E, \mathtt{r}, U, \lhd \rangle$, where:

- N is a set of nodes,
- $E \subseteq N \times \Sigma \times N$ is a set of edges labeled by input events,
- $U : N \to \mathsf{Form}^+(Q, \emptyset)$ is a labeling of nodes with positive sentences
- $\lhd \subseteq N \times N$ is a coverage relation,

initially `WorkList` $= \{\mathtt{r}\}$ and $N = E = U = \lhd = \emptyset$.

```
while WorkList $\neq \emptyset$ do
    dequeue $n$ from WorkList
    $N \leftarrow N \cup \{n\}$
    let $\alpha(n)$ be $a_1, \ldots, a_k$
    if $\overline{\Upsilon}(\alpha)(X^{(1)}, \ldots, X^{(k)})$ is satisfiable then          ▷ counterexample is feasible
        get model $\nu$ of $\overline{\Upsilon}(\alpha)(X^{(1)}, \ldots, X^{(k)})$
        return $w = (a_1, \nu(X^{(1)})) \ldots (a_k, \nu(X^{(k)}))$          ▷ $w \in L(\mathcal{A})$ by construction
    else                                                            ▷ spurious counterexample
        let $(I_0, \ldots, I_k)$ be a GLI for $\alpha$
        $b \leftarrow \bot$
        for $i = 0, \ldots, k$ do
            if $U(n_i) \not\models I_i$ then
                $Uncover \leftarrow \{m \in N \mid (m, n_i) \in \lhd\}$
                $\lhd \leftarrow \lhd \setminus \{(m, n_i) \mid m \in Uncover\}$          ▷ uncover the nodes covered by $n_i$
                for $m \in Uncover$ such that $m$ is a leaf of $\mathcal{U}$ do
                    enqueue $m$ into WorkList                       ▷ reactivate uncovered leaves
                $U(n_i) \leftarrow U(n_i) \wedge J_i$                    ▷ strenghten the label of $n_i$ (Lemma 7)
                if $\neg b$ then
                    $b \leftarrow$ Close$(n_i)$
    if $n$ is not covered then
        for $a \in \Sigma$ do                                                       ▷ expand $n$
            let $s$ be a fresh node and $e = (n, a, s)$ be a new edge
            $E \leftarrow E \cup \{e\}$
            $U \leftarrow U \cup \{(s, \top)\}$
            enqueue $s$ into WorkList
return $\top$
function Close$(x)$ returns $\mathbb{B}$
    for $y \in N$ such that $\alpha(y) \prec^* \alpha(x)$ do
        if $U(x) \models U(y)$ then
            $\lhd \leftarrow [\lhd \setminus \{(p, q) \in \lhd \mid q$ is $x$ or a successor of $x\}] \cup \{(x, y)\}$
            return $\top$
    return $\bot$
```

Definition 3. *Given an unfolding U of an automaton $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$ a node $\alpha \in$* $\mathrm{dom}(U)$ *is* covered *by another node $\beta \in$* $\mathrm{dom}(U)$*, denoted $\alpha \sqsubseteq \beta$, if and only if there exists a node $\alpha' \preceq \alpha$ such that $U(\alpha') \models U(\beta)$. Moreover, U is* closed *if and only if every leaf from* $\mathrm{dom}(U)$ *is covered by an uncovered node.*

A lazy annotation semi-algorithm will stop and report emptiness provided that it succeeds in building a closed and safe unfolding of the automaton. Notice that, by Definition 3, for any three nodes of an unfolding U, say $\alpha, \beta, \gamma \in \mathrm{dom}(U)$, if $\alpha \prec \beta$ and $\alpha \sqsubseteq \gamma$, then $\beta \sqsubseteq \gamma$ as well. As we show next (Theorem 2), there is no need to expand covered nodes, because, intuitively, there exists a word $w \in \mathcal{L}(\mathcal{A})$ such that $\alpha \preceq w_\Sigma$ and $\alpha \sqsubseteq \gamma$ only if there exists another word $u \in \mathcal{L}(\mathcal{A})$ such that $\gamma \preceq u_\Sigma$. Hence, exploring only those input event sequences that are continuations of γ (and ignoring those of α) suffices in order to find a counterexample for emptiness, if one exists.

An unfolding node $\alpha \in \mathrm{dom}(U)$ is said to be *spurious* if and only if $\Upsilon(\alpha)$ is unsatisfiable. In this case, we change (refine) the labels of (some of the) prefixes of α (and that of α), such that $U(\alpha)$ becomes $\bot$, thus indicating that there is no real execution of the automaton along that input event sequence. As a result of the change of labels, if a node $\gamma \preceq \alpha$ used to cover another node from $\mathrm{dom}(U)$, it might not cover it with the new label. Therefore, the coverage relation has to be recomputed after each refinement of the labeling. The semi-algorithm stops when (and if) a safe complete unfolding has been found.

Theorem 2. *If an automaton $\mathcal{A}$ has a nonempty safe closed unfolding then $\mathcal{L}(A) = \emptyset$.*

We describe the semi-algorithm used to check emptiness of first-order alternating automata. The execution of Algorithm 1 consists of three phases, corresponding to the Close, Refine and Expand of the original IMPACT procedure [20]. Let n be a node removed from the worklist at line 2 and let $\alpha(n)$ be the input sequence labeling the path from the root node to n. If $\overline{\Upsilon}(\alpha(n))$ is satisfiable, the sequence $\alpha(n)$ is feasible, in which case a model of $\overline{\Upsilon}(\alpha(n))$ is obtained and a word $w \in L(\mathcal{A})$ is returned. Otherwise, $\alpha(n)$ is an infeasible input sequence and the procedure enters the refinement phase (lines 9-19). The GLI for $\alpha(n)$ is used to strenghten the labels of all the ancestors of n, by conjoining the formulae of the interpolant, changed according to Lemma 7, to the existing labels.

In this process, the nodes on the path between $\mathtt{r}$ and n, including n, might become eligible for coverage, therefore we attempt to close each ancestor of n that is impacted by the refinement (line 19). Observe that, in this case the call to Close must uncover each node which is covered by a successor of n (line 30 of the Close function). This is required because, due to the over-approximation of the sets of reachable configurations, the covering relation is not transitive, as explained in [20]. If Close adds a covering edge (n_i, m) to $\lhd$, it does not have to be called for the successors of n_i on this path, which is handled via the boolean flag b. Finally, if n is still uncovered (it has not been previously covered during the refinement phase) we expand n (lines 21-25) by creating a new node for each successor s via the input event $a \in \Sigma$ and inserting it into the worklist.

4 Interpolant Generation

Typically, when checking the unreachability of a set of program configurations, the interpolants used to annotate the unfolded control structure are assertions about the values of the program variables in a given control state, at a certain step of an execution [20]. Because we consider alternating computation trees (forests), we must distinguish between (i) locality of interpolants w.r.t. a given control state (control locality) and

(ii) locality w.r.t. a given time stamp (time locality). In logical terms, *control-local* interpolants are formulae involving a single predicate symbol, whereas *time-local* interpolants involve only predicates $q^{(i)}$ and variables $x^{(i)}$, for a single $i \geq 0$. When considering alternating executions, control-local interpolants are not always enough to prove emptiness, because of the synchronization of several branches of the computation on the same input word. For this reason, the interpolants considered in this paper will never be control-local and we shall use the term *local* to denote time-local interpolants, with no free variables.

First, let us give the formal definition of the class of interpolants we shall work with. Given a formula ϕ, the *vocabulary* of ϕ, denoted $\mathrm{V}(\phi)$ is the set of predicate symbols $q \in Q^{(i)}$ and variables $x \in X^{(i)}$, occurring in ϕ, for some $i \geq 0$. For a term t, its vocabulary $\mathrm{V}(t)$ is the set of variables that occur in t. Observe that quantified variables and the interpreted function symbols of the data theory[3] do not belong to the vocabulary of a formula. By $\mathrm{P}^+(\phi)$ $[\mathrm{P}^-(\phi)]$ we denote the set of predicate symbols that occur in ϕ under an even [odd] number of negations.

Definition 4 ([19]). *Given formulae ϕ and ψ such that $\phi \wedge \psi$ is unsatisfiable, a* Lyndon interpolant *is a formula I such that $\phi \models I$, the formula $I \wedge \psi$ is unsatisfiable,* $\mathrm{V}(I) \subseteq \mathrm{V}(\phi) \cap \mathrm{V}(\psi)$*,* $\mathrm{P}^+(I) \subseteq \mathrm{P}^+(\phi) \cap \mathrm{P}^+(\psi)$ *and* $\mathrm{P}^-(I) \subseteq \mathrm{P}^-(\phi) \cap \mathrm{P}^-(\psi)$*.*

In the rest of this section, fix an automaton $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$. The following definition generalizes interpolants from unsatisfiable conjunctions to input sequences:

Definition 5. *Given a sequence of input events $\alpha = a_1 \ldots a_n \in \Sigma^*$, a* generalized Lyndon interpolant (GLI) *is a sequence $(I_0, \ldots, I_n)$ of formulae such that, for all $k \in [n-1]$, the following hold: (1) $\mathrm{P}^-(I_k) = \emptyset$, (2) $\iota^{(0)} \models I_0$, $I_k \wedge \Big(\bigwedge_{q(\mathbf{y}) \xrightarrow{a_i(X)} \psi \in \Delta} \forall y_1 \ldots \forall y_{\#(q)} \,.\, q^{(k)}(\mathbf{y}) \rightarrow \psi^{(k+1)} \Big) \models I_{k+1}$ and (3) $I_n \wedge \bigwedge_{q \in Q \setminus F} \forall y_1 \ldots \forall y_{\#(q)} \,.\, q(\mathbf{y}) \rightarrow \bot$ is unsatisfiable. Moreover, the GLI is* local *if and only if $\mathrm{V}(I_k) \subseteq Q^{(k)}$, for all $k \in [n]$.*

The following proposition states the existence of local GLI for the theories in which Lyndon's Interpolation Theorem holds.

Proposition 1. *If there exists a Lyndon interpolant for any two formulae ϕ and ψ, in the first-order theory of data with uninterpreted predicate symbols, such that $\phi \wedge \psi$ is unsatisfiable, then any sequence of input events $\alpha = a_1 \ldots a_n \in \Sigma^*$, such that $\Upsilon(\alpha)$ is unsatisfiable, has a local GLI $(I_0, \ldots, I_n)$.*

A problematic point of the above proposition is that the existence of Lyndon interpolants (Definition 4) is proved in principle, but the proof is non-constructive. In other words, the proof of Proposition 1 does not yield an algorithm for computing GLIs, for the following reason. Building an interpolant for an unsatisfiable conjunction of formulae $\phi \wedge \psi$ is typically the job of the decision procedure that proves the unsatisfiability and, in general, there is no such procedure, when ϕ and ψ contain predicates and have non-trivial quantifier alternation. In this case, some provers use instantiation heuristics for the universal quantifiers that are sufficient for proving unsatisfiability, however these

[3] E.g., the arithmetic operators of addition and multiplication, when $\mathbb{D}$ is the set of integers.

heuristics are not always suitable for interpolant generation. Consequently, from now on, we assume the existence of an effective Lyndon interpolation procedure only for decidable theories, such as the quantifier-free linear (integer) arithmetic with uninterpreted functions (UFLIA, UFLRA, etc.) [26].

This is where the predicate-free path formulae (defined in §2.1) come into play. Recall that, for a given event sequence α, the automaton $\mathcal{A}$ accepts a word w such that $w_\Sigma = \alpha$ if and only if $\overline{\Upsilon}(\alpha)$ is satisfiable (Lemma 5). Assuming further that the equality and interpreted predicates (e.g. inequalities for integers) atoms from the transition rules of $\mathcal{A}$ belong to a decidable first-order theory, such as Presburger arithmetic, Lemma 5 gives us an effective way of checking emptiness of $\mathcal{A}$, relative to a given event sequence. However, this method does not cope well with lazy annotation, because there is no way to extract, from the unsatisfiability proof of $\overline{\Upsilon}(\alpha)$, the interpolants needed to annotate α. This is because (I) the formula $\overline{\Upsilon}(\alpha)$, obtained by repeated substitutions loses track of the steps of the execution, and (II) quantifiers that occur nested in $\overline{\Upsilon}(\alpha)$ make it difficult to write $\overline{\Upsilon}(\alpha)$ as an unsatisfiable quantifier-free conjunction of formulae from which interpolants are extracted (Definition 4).

The solution we adopt for the first issue (I) consists in partially recovering the time-stamped structure of the acceptance formula $\Upsilon(\alpha)$ using the formula $\widehat{\Upsilon}(\alpha)$, in which only transition quantifiers occur. The second issue (II) is solved under the additional assumption that the theory of the data domain $\mathbb{D}$ has *witness-producing quantifier elimination*. More precisely, we assume that, for each formula $\exists x \,.\, \phi(x)$, there exists an effectively computable term τ, in which x does not occur, such that $\exists x \,.\, \phi$ and $\phi[\tau/x]$ are equisatisfiable. These terms, called *witness terms* in the following, are actual definitions of the Skolem function symbols from the following folklore theorem:

Theorem 3 ([3]). *Given $Q_1 x_1 \ldots Q_n x_n \,.\, \phi$ a first-order sentence, where $Q_1, \ldots, Q_n \in \{\exists, \forall\}$ and ϕ is quantifier-free, let $\eta_i \overset{\text{def}}{=} f_i(y_1, \ldots, y_{k_i})$ if $Q_i = \forall$ and $\eta_i \overset{\text{def}}{=} x_i$ if $Q_i = \exists$, where f_i is a fresh function symbol and $\{y_1, \ldots, y_{k_i}\} = \{x_j \mid j < i,\ Q_j = \exists\}$. Then the entailment $Q_1 x_1 \ldots Q_n x_n \,.\, \phi \models \phi[\eta_1/x_1, \ldots, \eta_n/x_n]$ holds.*

Examples of witness-producing quantifier elimination procedures can be found in the literature for e.g. linear integer (real) arithmetic (LIA,LRA), Presburger arithmetic and boolean algebra of sets and Presburger cardinality constraints (BAPA) [18].

Under the assumption that witness terms can be effectively built, we describe the generation of a non-local GLI for a given input event sequence $\alpha = a_1 \ldots a_n$. First, we generate successively the acceptance formula $\Upsilon(\alpha)$ and its equisatisfiable forms $\widehat{\Upsilon}(\alpha) = Q_1 x_1 \ldots Q_m x_m \,.\, \widehat{\Phi}$ and $\overline{\Upsilon}(\alpha) = Q_1 x_1 \ldots Q_m x_m \,.\, \overline{\Phi}$, both written in prenex form, with matrices $\widehat{\Phi}$ and $\overline{\Phi}$, respectively. Because we assumed that the first order theory of $\mathbb{D}$ has quantifier elimination, the satisfiability problem for $\overline{\Upsilon}(\alpha)$ is decidable. If $\overline{\Upsilon}(\alpha)$ is satisfiable, we build a counterexample for emptiness w such that $w_\Sigma = \alpha$ and $w_{\mathbb{D}}$ is a satisfying assignment for $\overline{\Upsilon}(\alpha)$. Otherwise, $\overline{\Upsilon}(\alpha)$ is unsatisfiable and there exist witness terms $\tau_{i_1} \ldots \tau_{i_\ell}$, where $\{i_1, \ldots, i_\ell\} = \{j \in [1,m] \mid Q_j = \forall\}$, such that $\overline{\Phi}[\tau_{i_1}/x_{i_1}, \ldots, \tau_{i_\ell}/x_{i_\ell}]$ is unsatisfiable (Theorem 3). Then it turns out that the formula $\widehat{\Phi}[\tau_{i_1}/x_{i_1}, \ldots, \tau_{i_\ell}/x_{i_\ell}]$, obtained analogously from the matrix of $\widehat{\Upsilon}(\alpha)$, is unsatisfiable as well (Lemma 6 below). Because this latter formula is structured as a conjunction of formulae $\iota^{(0)} \wedge \phi_1 \ldots \wedge \phi_n \wedge \psi$, where $\mathrm{V}(\phi_k) \cap Q^{(\le n)} \subseteq Q^{(k-1)} \cup Q^{(k)}$ and $\mathrm{V}(\psi) \cap Q^{(\le n)} \subseteq Q^{(n)}$, it is now possible to use an existing interpolation procedure for the quantifier-free theory of $\mathbb{D}$, extended with uninterpreted function symbols, to compute a (not necessarily local) GLI $(I_0,\ldots,I_n)$ such that $\mathrm{V}(I_k) \cap Q^{(\leq n)} \subseteq Q^{(k)}$, for all $k \in [n]$.

Example 3 (Contd. from Examples 1 *and* 2*).* The formula $\overline{\Upsilon}(\alpha)$ (Example 2) is unsatisfiable and let $\tau_2 \overset{\mathsf{def}}{=} z_1$ be the witness term for the universally quantified variable z_2. Replacing z_2 with τ_2 (z_1) in the matrix of $\widehat{\Upsilon}(\alpha)$ (Example 1) yields the unsatisfiable conjunction below, obtained after trivial simplifications:

$$\begin{aligned}&[z_1 \geq 0 \wedge q^{(0)}(z_1)] \wedge [q^{(0)}(z_1) \rightarrow x^{(1)} \geq 0 \wedge q^{(1)}(x^{(1)} + z_1)] \wedge \\ &\qquad [q^{(1)}(x^{(1)} + z_1) \rightarrow x^{(1)} + z_1 < 0 \wedge {q_f}^{(2)}(x^{(2)} + x^{(1)} + z_1)]\end{aligned}$$

A non-local GLI for the above conjunction is the sequence of formulae:

$$(q^{(0)}(z_1) \wedge z_1 \geq 0,\ x^{(1)} \geq 0 \wedge q^{(1)}(x^{(1)} + z_1) \wedge z_1 \geq 0,\ \bot) \qquad \blacksquare$$

We formalize and prove the correctness for the above construction of non-local GLI. A function $\xi : \mathbb{N} \rightarrow \mathbb{N}$ is *monotonic* iff for each $n < m$ we have $\xi(n) \leq \xi(m)$ and *finite-range* iff for each $n \in \mathbb{N}$ the set $\{m \mid \xi(m) = n\}$ is finite. If ξ is finite-range, we denote by $\xi^{-1}_{\max}(n) \in \mathbb{N}$ the maximal value m such that $\xi(m) = n$.

Lemma 6. *Given a non-empty input event sequence $\alpha = a_1 \ldots a_n \in \Sigma^*$, such that $\Upsilon(\alpha)$ is unsatisfiable, let $Q_1 x_1 \ldots Q_m x_m\ .\ \widehat{\Phi}$ be a prenex form of $\widehat{\Upsilon}(\alpha)$ and let $\xi : [1,m] \rightarrow [n]$ be a monotonic finite-range function mapping each transition quantifier to the minimal index from the sequence $\widehat{\Theta}(\alpha_0),\ldots,\widehat{\Theta}(\alpha_n)$ where it occurs. Then one can effectively build:*

1. *witness terms $\tau_{i_1},\ldots,\tau_{i_\ell}$, where $\{i_1,\ldots,i_\ell\} = \{j \in [1,m] \mid Q_j = \forall\}$ and $\mathrm{V}(\tau_{i_j}) \subseteq X^{(\leq \xi(i_j))} \cup \{x_k \mid k < i_j, Q_k = \exists\}$, $\forall j \in [1,\ell]$ such that $\widehat{\Phi}[\tau_{i_1}/x_{i_1},\ldots,\tau_{i_\ell}/x_{i_\ell}]$ is unsatisfiable, and*
2. *a GLI $(I_0,\ldots,I_n)$ for α, such that $\mathrm{V}(I_k) \subseteq Q^{(k)} \cup X^{(\leq k)} \cup \{x_j \mid j < \xi^{-1}_{\max}(k),\ Q_j = \exists\}$, for all $k \in [n]$.*

Consequently, under two assumptions about the first-order theory of the data domain, namely (i) witness-producing quantifier elimination, and (ii) Lyndon interpolation for the quantifier-free fragment with uninterpreted functions, we developed a generic method that produces GLIs for unfeasible input event sequences. Moreover, each formula in the interpolant refers only to the current predicate symbols, the current and past input variables and the existentially quantified transition variables introduced at the previous steps. The remaining questions are how to use these GLIs to label the sequences in the unfolding of an automaton (Definition 2) and compute coverage (Definition 3) between nodes of the unfolding.

4.1 Unfolding with Non-local Interpolants

As required by Definition 2, the unfolding U of an automaton $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$ is labeled by formulae $U(\alpha) \in \mathsf{Form}^+(Q,\emptyset)$, with no free symbols, other than predicate symbols, such that the labeling is compatible with the transition relation of the automaton. Each newly expanded input sequence of $\mathcal{A}$ is initially labeled with $\top$ and the labels are refined using GLIs computed from proofs of spuriousness. The following lemma describes the refinement of the labeling of an input sequence by a non-local GLI:

Lemma 7. *Let U be an unfolding of an automaton $\mathcal{A} = \langle \Sigma, X, Q, \iota, F, \Delta \rangle$ such that $\alpha = a_1 \ldots a_n \in \mathrm{dom}(U)$ and $(I_0, \ldots, I_n)$ is a GLI for α. Then the mapping $U' : \mathrm{dom}(U) \rightarrow \mathsf{Form}^+(Q, \emptyset)$ is an unfolding of $\mathcal{A}$, where:*

- *$U'(\alpha_k) = U(\alpha_k) \wedge J_k$, for all $k \in [n]$, where J_k is the formula obtained from I_k by removing the time stamp of each predicate symbol $q^{(k)}$ and existentially quantifying each free variable, and*
- *$U'(\beta) = U(\beta)$ if $\beta \in \mathrm{dom}(U)$ and $\beta \not\leq \alpha$,*

Moreover, α is safe in U'.

Observe that, by Lemma 6 (2), the set of free variables of a GLI formula I_k consists of (i) variables $X^{(\leq k)}$ keeping track of data values seen in the input at some earlier moment in time, and (ii) variables that track past choices made within the transition rules. Basically, it is not important when exactly in the past a certain input has been read or when a choice has been made, because only the relation between the values of these and the current variables determines the future behavior of the automaton. Quantifying these variables existentially does the job of ignoring when exactly in the past these values have been seen. Moreover, the last point of Lemma 7 ensures that the refined path is safe in the new unfolding and will stay safe in all future refinements of this unfolding.

The last ingredient of the lazy annotation semi-algorithm based on unfoldings consist in the implementation of the coverage check, when the unfolding of an automaton is labeled with conjunctions of existentially quantified formulae with predicate symbols, obtained from interpolation. By Definition 3, checking whether a given node $\alpha \in \mathrm{dom}(U)$ is covered amounts to finding a prefix $\alpha' \leq \alpha$ and a node $\beta \in \mathrm{dom}(U)$ such that $U(\alpha') \models U(\beta)$, or equivalently, the formula $U(\alpha') \wedge \neg U(\beta)$ is unsatisfiable. However, the latter formula, in prenex form, has quantifier prefix in the language $\exists^* \forall^*$ and, as previously mentioned, the satisfiability problem for such formulae becomes undecidable when the data theory subsumes Presburger arithmetic [10].

Nevertheless, if we require just a yes/no answer (i.e. not an interpolant) recently developed quantifier instantiation heuristics [25] perform rather well in answering a large number of queries in this class. Observe, moreover, that coverage does not need to rely on a complete decision procedure. If the prover fails in answering the above satisfiability query, then the semi-algorithm assumes that the node is not covered and continues exploring its successors. Failure to compute complete coverage may lead to divergence (non-termination) and ultimately, to failure to prove emptiness, but does not affect the soundness of the semi-algorithm (real counterexamples will still be found).

5 Experimental Results

We have implemented a version of the IMPACT semi-algorithm [20] in a prototype tool, avaliable online [8]. The tool is written in Java and uses the Z3 SMT solver [27], via the JavaSMT interface [15], for spuriousness and coverage queries and also for interpolant generation. Table 1 reports the size of the input automaton in bytes, the numbers of Predicates, Variables and Transitions, the result of emptiness check, the number of Expanded and Visited Nodes during the unfolding and the Time in miliseconds. The experiments were carried out on a MacOS x64 - 1.3 GHz Intel Core i5 - 8 GB 1867 MHz LPDDR3 machine.

Example	$\|\mathcal{A}\|$ (bytes)	Predicates	Variables	Transitions	$L(\mathcal{A}) = \emptyset$?	Nodes Expanded	Nodes Visited	Time (msec)
incdec.pa	499	3	1	12	no	21	17	779
localdec.pa	678	4	1	16	no	49	35	1814
ticket.pa	4250	13	1	73	no	229	91	9543
count_thread0.pa	9767	14	1	126	no	154	128	8553
count_thread1.pa	10925	15	1	135	no	766	692	76771
local0.pa	10595	13	1	117	no	73	27	1431
local1.pa	11385	14	1	126	no	1135	858	101042
array_rotation.ada	1834	8	7	7	yes	9	8	1543
array_simple.ada	3440	9	5	8	yes	11	10	6787
array_shift.ada	874	6	5	5	yes	6	5	413
abp.ada	6909	16	14	28	no	52	47	4788
train.ada	1823	10	4	26	yes	68	67	7319
hw1.ada	322	3	2	5	Solver Error	/	/	/
hw2.ada	674	7	2	8	yes	20	22	4974
rr-crossing.foada	1780	10	1	16	yes	67	67	7574
train-simple1.foada	5421	13	1	61	yes	43	44	2893
train-simple2.foada	10177	16	1	118	yes	111	113	8386
train-simple3.foada	15961	19	1	193	yes	196	200	15041
fischer-mutex2.foada	3000	11	2	23	yes	23	23	808
fischer-mutex3.foada	4452	16	2	34	yes	33	33	1154

Table 1. Experiments with First Order Alternating Automata

The test cases shown in Table 1, come from several sources, namely predicate automata models (*.pa) [6,7] available online [23], timed automata inclusion problems (`abp.ada`, `train.ada`, `rr-crossing.foada`), array logic entailments (`array_rotation.ada`, `array_simple.ada`, `array_shift.ada`) and hardware circuit verification (`hw1.ada`, `hw2.ada`), initially considered in [13], with the restriction that local variables are made visible in the input. The `train-simpleN. foada` and `fischer-mutexN. foada` examples are parametric verification problems in which one checks inclusions of the form $\bigcap_{i=1}^{N} \mathcal{L}(A_i) \subseteq \mathcal{L}(B)$, where A_i is the i-th copy of the template automaton.

The advantage of using FOADA over the INCLUDER [12] tool from [13] is the possibility of having automata over infinite alphabets with local variables, whose values are not visible in the input. In particular, this is essential for checking inclusion of timed automata that use internal clocks to control the computation.

6 Conclusions

We present first-order alternating automata, a model of computation that generalizes classical boolean alternating automata to first-order theories. Due to their expressivity, first-order alternating automata are closed under union, intersection and complement. However the emptiness problem is undecidable even in the most simple case, of the quantifier-free theory of equality with uninterpreted predicate symbols. We deal with the emptiness problem by developping a practical semi-algorithm that always terminates, when the automaton is not empty. In case of emptiness, termination of the semi-algorithm occurs in most practical test cases, as shown by a number of experiments.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
2. Barringer, H., Rydeheard, D., Havelund, K.: Rule systems for run-time monitoring: From eagle to ruler. In: Sokolsky, O., Taşıran, S. (eds.) Runtime Verification. pp. 111–125. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

3. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Perspectives in Mathematical Logic, Springer (1997)
4. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)
5. D'Antoni, L., Kincaid, Z., Wang, F.: A symbolic decision procedure for symbolic alternating finite automata. Electronic Notes in Theoretical Computer Science 336, 79 – 99 (2018), the Thirty-third Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIII)
6. Farzan, A., Kincaid, Z., Podelski, A.: Proof spaces for unbounded parallelism. SIGPLAN Not. 50(1), 407–420 (Jan 2015)
7. Farzan, A., Kincaid, Z., Podelski, A.: Proving liveness of parameterized programs. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science. pp. 185–196. LICS '16, ACM (2016)
8. First Order Alternating Data Automata (FOADA). https://github.com/cathiec/FOADA
9. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. SIGPLAN Not. 47(6), 405–416 (Jun 2012)
10. Halpern, J.Y.: Presburger arithmetic with unary predicates is π_1^1 complete. The Journal of Symbolic Logic 56(2), 637–642 (1991)
11. Hojjat, H., Rümmer, P.: Deciding and interpolating algebraic data types by reduction (technical report). CoRR abs/1801.02367 (2018), HTTP://ARXIV.ORG/ABS/1801.02367
12. Includer. http://www.fit.vutbr.cz/research/groups/verifit/tools/includer/
13. Iosif, R., Rogalewicz, A., Vojnar, T.: Abstraction refinement and antichains for trace inclusion of infinite state systems. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2016). pp. 71–89 (2016)
14. Iosif, R., Xu, X.: Abstraction refinement for emptiness checking of alternating data automata. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2018). pp. 93–111 (2018)
15. JavaSMT. https://github.com/sosy-lab/java-smt
16. Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science 134(2), 329 – 363 (1994)
17. Kincaid, Z.: Parallel Proofs for Parallel Programs. Ph.D. thesis, University of Toronto (2016)
18. Kuncak, V., Mayer, M., Piskac, R., Suter, P.: Software synthesis procedures. Commun. ACM 55(2), 103–111 (2012)
19. Lyndon, R.C.: An interpolation theorem in the predicate calculus. Pacific J. Math. 9(1), 129–142 (1959)
20. McMillan, K.L.: Lazy abstraction with interpolants. In: Proc. of CAV'06. LNCS, vol. 4144. Springer (2006)
21. McMillan, K.L.: Lazy annotation revisited. In: CAV2014, Proceedings. pp. 243–259. Springer International Publishing (2014)
22. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM 27(2), 356–364 (Apr 1980)
23. Predicate Automata. https://github.com/zkincaid/duet/tree/ark2/regression/predicateAutomata
24. Presburger, M.: Über die Vollstandigkeit eines gewissen Systems der Arithmetik. Comptes rendus du I Congrés des Pays Slaves (Warsaw 1929)
25. Reynolds, A., King, T., Kuncak, V.: Solving quantified linear arithmetic by counterexample-guided instantiation. Formal Methods in System Design 51(3), 500–532 (2017)
26. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. J. Symb. Comput. 45(11), 1212–1233 (2010)
27. Z3 SMT Solver. https://rise4fun.com/z3