First-Order Automata

Luca Geatti[1], **Alessandro Gianola**[2], **Nicola Gigante**[3]

[1]Dipartimento di Scienze Matematiche, Informatiche e Fisiche, University of Udine, Udine, Italy
[2]INESC-ID/Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
[3]Faculty of Engineering, Free University of Bozen-Bolzano, Bolzano, Italy
luca.geatti@uniud.it, alessandro.gianola@tecnico.ulisboa.pt, nicola.gigante@unibz.it

Abstract

First-order linear temporal logic (FOLTL) is a flexible and expressive formalism capable of naturally describing complex behaviors and properties. Although the logic is in general highly undecidable, the idea of using it as a specification language for the verification of complex infinite-state systems is appealing. However, a missing piece, which has proved to be an invaluable tool in dealing with other temporal logics, is an *automaton* model capable of capturing the logic. In this paper we address this issue, by defining and studying such a model, which we call *first-order automaton*. We define this very general class of automata, and the corresponding notion of *regular* first-order language (of *finite* words), showing their closure under most language-theoretic operations. We show how they can capture any FOLTL formula over finite words, over any signature and theory, and provide sufficient conditions for the *semi-decidability* of their non-emptiness problem. Then, to show the usefulness of the formalism, we prove the decidability of *monodic* FOLTL, a classic result known in the literature, with a simpler and direct proof.

1 Introduction

Linear temporal logic (LTL) (Pnueli 1977) is the most common specification language for the verification of reactive systems. LTL on *finite words* ($\mathsf{LTL_f}$, (De Giacomo and Vardi 2013)) also gained traction recently in many applications in AI and formal verification. However, LTL and $\mathsf{LTL_f}$ are *propositional* logics, and as such they are limited to the specification of *finite-state* systems, whereas many real-world scenarios require modeling an infinite state space.

First-order LTL (FOLTL) is a temporal logic that mixes LTL with classical first-order logic, resulting in an extremely expressive and flexible formalism. The logic has been studied quite extensively in the literature (Gabbay et al. 2003; Kontchakov et al. 2004), especially in the field of *knowledge representation*, where it is used as the formal underpinning of temporal description logics, temporal databases, *etc*. However, reasoning about FOLTL is quite hard: in general, satisfiability of FOLTL sentences is highly undecidable, *i.e.*, not even recursively enumerable (Gabbay et al. 2003). Nevertheless, the idea of using well-behaved fragments of FOLTL as a specification language for verification of infinite-state systems is appealing and worth investigating. We are interested in tasks such as satisfiability and model checking (Clarke et al. 2018) of FOLTL specifications, but also *reactive synthesis* (Pnueli and Rosner 1989), *i.e.*, building a *strategy* that fulfils a FOLTL specification in adversarial environments, and *monitoring*, *i.e.*, detecting violations of a specification during a system's execution (Leucker and Schallhart 2009). Indeed, the MoXI intermediate language for model checking features a semantics based on FOLTL (Johannsen et al. 2024; Rozier et al. 2024), specifically to support infinite-state model checking.

However, to make FOLTL a really practicable specification language, we miss some ingredients. One of these is the definition of a model of *automata* capable of capturing FOLTL, and a theory of the *languages* (sets of models) that FOLTL can describe. These ingredients have been crucial in the development of the theory and practice of finite-state verification, with formal language theory being the theoretical background (Loeckx and Sieber 1987) and automata providing the algorithmic powerhouse (Vardi 2005, 1991).

In this paper, we define and study a new kind of infinite-state, finitely representable automata on *finite words*, that we call *first-order automata*, capable of capturing FOLTL on finite words. In particular, our contributions are the following:

1. we introduce the definition of first-order automata and the associated notion of *first-order regular languages*;
2. we study the *closure properties* of these languages, *i.e.*, that they are closed by union, intersection, concatenation, and Kleene star, and we discuss deterministic first-order automata (although determinization is still open);
3. we show how first-order automata can easily and directly encode any FOLTL sentence over any first-order theory;
4. we show, similarly to what happens for propositional LTL, that if we start with an FOLTL sentence that uses only *past operators*, the corresponding automaton is *deterministic*; as determinism is essential to solve *reactive synthesis* and *monitoring* for LTL specifications (Artale et al. 2023), this result is encouraging;
5. we address the non-emptiness problem for first-order automata (*i.e.*, the existence of any accepted word), providing some sufficient conditions for its *semi-decidability*;
6. we discuss some sufficient conditions for *decidability* of the emptiness problem, including a class of *monadic*

automata which are the natural counterpart of *monodic* FOLTL sentences (*i.e.*, the fragment where any temporal subformula has only one free variable).

As evidence of the usefulness of automata in dealing with FOLTL, Item 6 provides a self-contained and more elementary proof of the decidability of *monodic* FOLTL on finite words. This is a classic result (Gabbay et al. 2003) whose original proof, however, is conceptually quite complex, since it relies on the decidability result of monadic second order logic over $(\mathbb{N}, <)$ by Büchi (Büchi 1962).

We proceed as follows. Section 2 provides some background, and Section 3 defines first-order automata and first-order regular languages, studying their closure properties. Section 4 shows how first-order automata capture FOLTL, and Section 5 shows how pure-past sentences result in deterministic automata, discussing the implications of this construction in applications. Section 6 discusses the non-emptiness problem, and Section 7 discusses some decidable cases, including our self-contained proof of the decidability of monodic FOLTL. Section 8 concludes with a discussion and related work. All the proofs can be found in the extended version of the paper (Geatti, Gianola, and Gigante 2024).

2 Preliminaries

First-Order Logic and Languages. We adopt standard *first-order logic* notions, considering *multi-sorted* (first-order) *signatures* and (first-order) *structures* over a signature Σ (Σ-structures). We assume a fixed set of sort symbols S_1, S_2, *etc.* As usual, given a (constant, function, or relation) symbol s from Σ and a Σ-structure σ, we denote as s^σ the interpretation of s given by σ, and similarly for sorts. Given a first-order signature Σ, a Σ-theory $\mathcal{T}$ is a set of sentences over Σ (Σ-sentences). Given a first-order signature Σ and a Σ-theory $\mathcal{T}$, we denote as $[\![\mathcal{T}]\!]$ the set of all the *finite* or *countably infinite* Σ-structures σ such that $\sigma \models \mathcal{T}$.

We assume that signatures mark each symbol to be either *rigid* or *non-rigid*. Given a signature Σ, we denote as Σ' the signature obtained from Σ by replacing any *non-rigid* symbol s with a primed version s'. Similarly, given a Σ-structure σ we denote as σ' the corresponding renamed Σ'-structure. Furthermore, given a formula ϕ, we denote as $\phi[\Sigma/\Sigma']$ the formula obtained by replacing any symbol from Σ with the corresponding primed symbol of Σ'. The notation Σ'' *etc.* has similar meaning. We define first-order words as follows.

Definition 1 (First-order word)**.** *Let* Σ *be a* signature *and* $\mathcal{T}$ *a* Σ*-theory. A* first-order word *modulo* $\mathcal{T}$ *(*i.e.*, a* $\mathcal{T}$*-word) is a finite sequence* $\overline{\sigma} = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle$ *of first-order structures* $\sigma_i \in [\![\mathcal{T}]\!]$ *such that for each* sort S *and each* rigid *symbol* $s \in \Sigma$*, we have* $S^{\sigma_i} = S^{\sigma_j}$ *and* $s^{\sigma_i} = s^{\sigma_j}$ *for all* $0 \le i, j < n$*.*

Intuitively, $\mathcal{T}$-words are sequences of $\mathcal{T}$-structures, where the interpretation of *rigid* symbols is constant throughout the word. Note that the domain of each sort is fixed throughout the word, so we are working in a *constant domains* setting. First-order words are just called *words* when there is no ambiguity. We assume $\mathcal{T} = \varnothing$ when not specified. The *length* of a word $\overline{\sigma} = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle$ is denoted as $|\overline{\sigma}| = n$. We denote with $[\![\mathcal{T}]\!]^*$ and $[\![\mathcal{T}]\!]^+$ the set of all and all non-empty $\mathcal{T}$-words, respectively.

Definition 2 (First-order languages)**.** *Given a signature* Σ *and a* Σ*-theory* $\mathcal{T}$*, a* first-order $\mathcal{T}$-language $\mathcal{L} \subseteq [\![\mathcal{T}]\!]^*$ *is a set of* $\mathcal{T}$*-words.*

First-order linear temporal logic. FOLTL is a first-order temporal logic that blends LTL and first-order logic. The syntax can be described as follows. Let Σ be a signature. A Σ-term is defined as follows:

$$t \coloneqq c \mid x \mid f(t_1, \ldots, t_n)$$

where c is a constant, x is a first-order variable, f is an n-ary function symbol, and $t_1, \ldots, t_n$ are Σ-terms. The syntax of FOLTL formulas is defined as follows:

$$\begin{aligned} \phi \coloneqq\ & p(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \neg\phi \mid \phi \vee \phi \mid \exists x \phi \\ & \mid \mathsf{X}\phi \mid \mathsf{Y}\phi \mid \phi \,\mathsf{U}\, \phi \mid \phi \,\mathsf{S}\, \phi \end{aligned}$$

where $t_1, \ldots, t_n$ are Σ-terms, p is an n-ary predicate symbol of Σ (where n can also be 0, having atomic proposition), and x is a first-order variable. The temporal operators $\mathsf{X}, \mathsf{Y}, \mathsf{U}, \mathsf{S}$ are called *tomorrow*, *yesterday*, *until*, and *since* respectively. We point out that classic first-order logic formulas correspond to FOLTL formulas ϕ without temporal operators.

Σ-terms are evaluated on Σ-structures. Given a Σ-term t and a Σ-structure σ, an *environment* is a mapping ξ from any first-order variables of sort S to values of their respective domain S^σ. Then, the *evaluation* of t on σ with environment ξ, denoted $t^{\sigma,\xi}$ is defined in a standard way, that is, $c^{\sigma,\xi} = c^\sigma$, $x^{\sigma,\xi} = \xi(x)$, and $f(t_1, \ldots, t_n)^{\sigma,\xi} = f^\sigma(t_1^{\sigma,\xi}, \ldots, t_n^{\sigma,\xi})$, where c is a constant, x is a first-order variable, f is an n-ary function symbol, and $t_1, \ldots, t_n$ are Σ-terms.

Formulas of FOLTL are interpreted over non-empty first-order words. Let ϕ be a FOLTL formula and let $\overline{\sigma} \in [\![\mathcal{T}]\!]^+$. The *satisfaction* of ϕ by $\overline{\sigma}$, with environment ξ, at time i, denoted $\overline{\sigma}, \xi, i \models \phi$, is defined as follows:

- $\overline{\sigma}, \xi, i \models p(t_1, \ldots, t_n)$ iff $(t_1^{\sigma_i,\xi}, \ldots, t_n^{\sigma_i,\xi}) \in p^{\sigma_i}$;
- $\overline{\sigma}, \xi, i \models t_1 = t_2$ iff $t_1^{\sigma_i,\xi} = t_2^{\sigma_i,\xi}$;
- $\overline{\sigma}, \xi, i \models \neg\phi$ iff $\overline{\sigma}, \xi, i \not\models \phi$;
- $\overline{\sigma}, \xi, i \models \phi_1 \vee \phi_2$ iff $\overline{\sigma}, \xi, i \models \phi_1$ or $\overline{\sigma}, \xi, i \models \phi_2$;
- $\overline{\sigma}, \xi, i \models \exists x \phi$ iff $\overline{\sigma}, \xi', i \models \phi$ *for some* ξ' agreeing with ξ except possibly for x;
- $\overline{\sigma}, \xi, i \models \mathsf{X}\phi$ iff $i < |\overline{\sigma}| - 1$ and $\overline{\sigma}, \xi, (i+1) \models \phi$;
- $\overline{\sigma}, \xi, i \models \mathsf{Y}\phi$ iff $i > 0$ and $\overline{\sigma}, \xi, (i-1) \models \phi$;
- $\overline{\sigma}, \xi, i \models \phi_1 \,\mathsf{U}\, \phi_2$ iff there exists a $k \ge i$ such that $\overline{\sigma}, \xi, k \models \phi_2$ and $\overline{\sigma}, \xi, j \models \phi_1$ for all $i \le j < k$;
- $\overline{\sigma}, \xi, i \models \phi_1 \,\mathsf{S}\, \phi_2$ iff there exists a $0 \le k \le i$ such that $\overline{\sigma}, \xi, k \models \phi_2$ and $\overline{\sigma}, \xi, j \models \phi_1$ for all $k < j \le i$.

We define the following shortcuts: (i) $\widetilde{\mathsf{X}}\phi \equiv \neg\mathsf{X}\neg\phi$, called *weak tomorrow*; (ii) $\mathsf{Z}\phi \equiv \neg\mathsf{Y}\neg\phi$, called *weak yesterday*; (iii) $\phi_1 \,\mathsf{R}\, \phi_2 \equiv \neg(\neg\phi_1 \,\mathsf{U}\, \neg\phi_2)$, called *release*; (iv) $\phi_1 \,\mathsf{T}\, \phi_2 \equiv \neg(\neg\phi_1 \,\mathsf{S}\, \neg\phi_2)$, called *triggered.* We point out that, for any FOLTL formula ϕ, $\widetilde{\mathsf{X}}\phi$ (resp., $\mathsf{Z}\phi$) is true at the last (resp., first) time point of a first-order word. The operators $\mathsf{X}, \widetilde{\mathsf{X}}, \mathsf{U}, \mathsf{R}$ are called *future* temporal operators, while $\mathsf{Y}, \mathsf{Z}, \mathsf{S}, \mathsf{T}$ are called *past* temporal operators.

Given a *sentence* ϕ (*i.e.*, a formula with no free variables), we say that $\overline{\sigma}$ *satisfies* ϕ, denoted $\overline{\sigma} \models \phi$, if $\overline{\sigma}, \xi, 0 \models \phi$ for any ξ. The $\mathcal{T}$-language of an FOLTL sentence ϕ, denoted $\mathcal{L}_{\mathcal{T}}(\phi)$, is the set of all the $\mathcal{T}$-words $\overline{\sigma} \in [\![\mathcal{T}]\!]^+$ that satisfy ϕ. A FOLTL sentence ϕ is *satisfiable* iff $\mathcal{L}_{\mathcal{T}}(\phi) \neq \varnothing$. Note that FOLTL semantics subsumes that of classical first-order logic. A first-order formula ϕ is satisfied by a Σ-structure σ iff $\overline{\sigma} \models \phi$ where $\overline{\sigma} = \langle \sigma \rangle$, and this corresponds to the classical first-order semantics.

FOLTL satisfiability, *i.e.*, the problem of establishing whether a FOLTL formula is satisfiable, is known to be undecidable, in general. See (Gabbay et al. 2003) for a full treatment where, however, constants and relational symbols are always assumed to be rigid and non-rigid, respectively.

3 First-Order Automata

A *first-order automaton* is a device that accepts first-order languages. A naive definition would result into an infinite-state representation not suitable for being handled computationally. Instead, here we make use of a *symbolic* representation, which exploits first-order logic to represent such infinite objects finitarily.

Definition 3 (First-order automata). *A* first-order automaton *is a tuple $\mathcal{A} = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ where:*

- *Σ is a* finite *signature called the* word signature*;*
- *Γ is a* finite *signature,* disjoint *from Σ, called the* state signature*;*
- *ϕ_0 is a first-order* Γ-sentence *called the* initial condition*;*
- *ϕ_T is a first-order* $(\Gamma \cup \Sigma \cup \Gamma')$-sentence *called the* transition relation*;*
- *ϕ_F is a first-order* Γ-sentence *called the* acceptance condition.

Intuitively, an automaton has an infinite state space made of Γ-structures, which are called *states*, with a transition relation represented symbolically by ϕ_T. We say only *automaton* to mean a first-order automaton as in Definition 3, when there is no ambiguity. Automata induce *runs*, *i.e.*, sequences of states, when *reading* a Σ-word.

Definition 4 (Run). *Let $\mathcal{A} = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ be an automaton, and let $\overline{\sigma} = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle$ be $\mathcal{T}$-word for some Σ-theory $\mathcal{T}$. A $\mathcal{T}$-run of $\mathcal{A}$ induced by $\overline{\sigma}$ is an $\varnothing$-word $\overline{\rho} = \langle \rho_0, \ldots, \rho_n \rangle$ over Γ such that $S^{\rho_i} = S^{\sigma_0}$ for any $0 \leq i \leq n$ and any sort symbol S, $\rho_0 \models \phi_0$ and $(\rho_i \cup \sigma_i \cup \rho'_{i+1}) \models \phi_T$ for all $0 \leq i < n$.*

Note that automata are *nondeterministic*, hence the same word induces more than one run, in general. The acceptance condition of first-order automata is a standard reachability condition, defined as follows.

Definition 5 (Acceptance of words). *Given an automaton $\mathcal{A} = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$, let $\mathcal{T}$ be a Σ-theory, and let $\overline{\sigma} = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle$ be a $\mathcal{T}$-word. We say that $\mathcal{A}$ $\mathcal{T}$-*accepts *$\overline{\sigma}$ iff there exists a $\mathcal{T}$-run $\overline{\rho} = \langle \rho_0, \ldots, \rho_n \rangle$ such that $\rho_n \models \phi_F$.*

The *$\mathcal{T}$-language* of an automaton $\mathcal{A}$, denoted $\mathcal{L}_{\mathcal{T}}(\mathcal{A})$ is the set of all the $\mathcal{T}$-words $\mathcal{T}$-accepted by $\mathcal{A}$. Then, *first-order regular languages* are defined consequently.

Definition 6 (First-order $\mathcal{T}$-regular language). *Given a Σ-theory $\mathcal{T}$, a $\mathcal{T}$-language $\mathcal{L} \subseteq [\![\mathcal{T}]\!]^*$ is said to be* $\mathcal{T}$-regular *iff there is a first-order automaton $\mathcal{A}$ such that $\mathcal{L}_{\mathcal{T}}(\mathcal{A}) = \mathcal{L}$.*

Example 1. *Consider a signature $\Sigma = \{c, >, +, \cdot\}$ where c is a non-rigid constant and $>$, $+$, and $\cdot$ are the standard (rigid) arithmetic symbols. Let $\mathcal{T}$ be the standard theory of integer arithmetics. Let $\mathcal{A} = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ where*

1. *$\Gamma = \{a, n\}$ with a and n two constants;*
2. *$\phi_0 := n = 0$;*
3. *$\phi_T := n' = n + 1 \wedge (n+1) \cdot a' = n \cdot a + c$; and*
4. *$\phi_F := a = 0$.*

It can be seen that $\mathcal{A}$ accepts all the $\mathcal{T}$-words such that the average of the values of c over all the word is zero.

Basic properties of $\mathcal{T}$-regularity. Let us now study some basic questions about $\mathcal{T}$-regular languages. As a first step, some cardinality arguments allow us to conclude that, despite the evident expressive power of the formalism, there are $\mathcal{T}$-languages, for some $\mathcal{T}$, that cannot be accepted by any first-order automaton.

Theorem 1. *For some signature Σ and Σ-theory $\mathcal{T}$, there are $\mathcal{T}$-languages that are not $\mathcal{T}$-regular.*

Despite Theorem 1, a concrete example of non $\mathcal{T}$-regular language has still not been found. Now, we can show that $\mathcal{T}$-regular languages are closed under some common operations. We start with union and intersection.

Theorem 2. *For any theory $\mathcal{T}$, $\mathcal{T}$-regular languages are closed under intersection and union.*

To look at concatenation and Kleene star, we need to investigate more closely the expressive power of first-order automata. It turns out they leak towards second-order territory.

Example 2. *Consider the signature $\Sigma = \{R, s, d\}$ with R a binary predicate and s and d two constants. It is well-known that one cannot write a first-order formula ϕ that holds in a Σ-structure if and only if d is* not *R-reachable from s. However, consider the language $\mathcal{L}$ made of $\varnothing$-words $\overline{\sigma}$ over Σ where each element σ of the word is such that d^σ is* not *R^σ-reachable from s^σ. We can build a first-order automaton $\mathcal{A} = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ that accepts $\mathcal{L}$, as follows. We define $\Gamma = \{P\}$ where P is a unary predicate, and $\phi_0 \equiv \phi_f \equiv \top$. Then, we define ϕ_T as follows:*

$$\phi_T \equiv P(s) \wedge \forall xy((P(x) \wedge R(x,y)) \rightarrow P(y)) \wedge \neg P(d)$$

Intuitively, the additional predicate P provided by Γ acts as the fixpoint of R. Indeed, the second-order logic formula $\exists P \ . \ \phi_T$ is exactly the one expressing non-reachability between s and d in Σ.

Example 2 shows that there are non-first-order-definable properties that can be enforced on single $\mathcal{T}$-structures. In general, we can express *existential* second-order properties.

Theorem 3. *Let Σ be a first-order signature, and $\mathcal{T}$ a Σ-theory. Let $\mathcal{C}$ be a class of Σ-structures, and let $\mathcal{L}_{\mathcal{C}}$ be the language of all the $\mathcal{T}$-words whose letters belong to $\mathcal{C}$. If $\mathcal{C}$ is definable by an* existential second-order logic *Σ-sentence, then $\mathcal{L}_{\mathcal{C}}$ is $\mathcal{T}$-regular.*

Intuitively, the existence of the run, required for accepting the word, performs the role of the existential second-order quantifiers. Note that this means that one may directly use existential second-order sentences as transition relations.

Definition 7 (Σ^1_1-automata)**.** *An* Σ^1_1-automaton *is a tuple $\mathcal{A}_1 = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ similar to a first-order automaton except that ϕ_0, ϕ_T, and ϕ_F are* existential second-order *sentences. Acceptance of words resembles Definition 5.*

Lemma 1. *Σ^1_1-automata are equivalent to first-order ones.*

The syntactic sugar of Σ^1_1-automata helps us proving further closure properties of $\mathcal{T}$-regular languages.

Theorem 4. *$\mathcal{T}$-regular languages are closed under concatenation, for any $\mathcal{T}$.*

Theorem 5. *$\mathcal{T}$-regular languages are closed under Kleene star, for any $\mathcal{T}$.*

Therefore, $\mathcal{T}$-regular languages are closed under the usual language-theoretic operations of regular expressions, which justifies the term *regular*. Whether they are closed under complementation as well is still an open question. The usual path is complementing an automaton by first determinizing it, therefore a preliminary step for complementation is that of defining determinism and deterministic automata.

However, determinism is more difficult to define in symbolically represented automata such as first-order ones, because the transition relation is not explicitly defined. In principle, to define what it means for a first-order automaton to be *deterministic*, one would want to force the transition relation to be a function, that is, when fixed a source state and a letter, the destination state would be unique. However, this is a too strong requirement for a first-order formula, given the Lowenheim-Skolem theorem.[1] Therefore, we only ask for all the destination states to be elementarily equivalent.

In the following, given a structure μ over a signature Σ_1, and a subsignature $\Sigma_2 \subseteq \Sigma_1$, we denote as $\mu|\Sigma_2$ the substructure of μ over Σ_2.

Definition 8 (Deterministic first-order automaton)**.** *Consider a first-order automaton $\mathcal{A} = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ and $\mathcal{T}$ be a Σ-theory. Then, $\mathcal{A}$ is said to be $\mathcal{T}$-*deterministic *iff:*

1. *ϕ_0 is satisfied by only one Γ-structure up to elementary equivalence;*
2. *for each two structures μ_1 and μ_2 such that both $\mu_1 \models \phi_T$ and $\mu_2 \models \phi_T$, if $\mu_1|_\Gamma$ and $\mu_2|_\Gamma$ are* elementarily equivalent *and so are $\mu_1|_\Sigma$ and $\mu_2|_\Sigma$, then $\mu_1|_{\Gamma'}$ and $\mu_2|_{\Gamma'}$ are elementarily equivalent as well.*

A feature that will often come useful together with determinism is *completeness*, that is, the absence of states without successors. An automaton $\mathcal{A} = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ is *complete* if, for all $(\Gamma \cup \Sigma)$-structures μ, there exists a Γ'-structure μ' such that $\mu \cup \mu' \models \phi_T$.

In finite-state automata, a straightforward consequence of determinism and completeness is that the run of a given word exists and is unique. Here we cannot, again, ask for uniqueness, but for elementary equivalence. The following is easy to show but confirms that Definition 8 is sufficient for our purposes.

[1]Given any transition between Γ-structures of infinite cardinalities, there will always exist another transition between Γ-structures of higher infinite cardinalities.

Proposition 1. *For any* deterministic *and* complete *first-order automaton $\mathcal{A}$ and $\mathcal{T}$-word $\overline{\sigma}$, the run of $\overline{\sigma}$ over $\mathcal{A}$ exists and is unique up to element-wise elementary equivalence.*

Therefore, if determinization were possible, then $\mathcal{T}$-regular languages would be closed under complementation, because deterministic first-order automata are easily complementable.

Proposition 2. *From a* deterministic *first-order automaton $\mathcal{A}$ one can obtain a complement automaton $\mathcal{A}^{-1}$ such that $\mathcal{L}_\mathcal{T}(\mathcal{A}^{-1}) = \overline{\mathcal{L}_\mathcal{T}(\mathcal{A})}$ for any theory $\mathcal{T}$.*

4 From Temporal Logic to Automata

We can now state how to encode FOLTL sentences into first-order automata. In all what follows, *w.l.o.g.* we assume any sentence to be in *negated normal form*, *i.e.*, with negations only applied to atoms as in $\neg p(t_1, \ldots, t_n)$ or $\neg(t_1 = t_n)$.

The translation first involves the following normal form.

Definition 9 (Stepped normal form)**.** *Given an* FOLTL *formula ϕ, the* stepped normal form *of ϕ, denoted* $\mathrm{snf}(\phi)$*, is the formula defined recursively as follows:*

1. *$\mathrm{snf}(p(t_1, \ldots, t_n)) = p(t_1, \ldots, t_n)$ and $\mathrm{snf}(t_1 = t_2) = (t_1 = t_2)$;*
2. *$\mathrm{snf}(Qx\phi) = Qx\,\mathrm{snf}(\phi)$, where $Q \in \{\forall, \exists\}$;*
3. *$\mathrm{snf}(\neg\phi) = \neg\,\mathrm{snf}(\phi)$;*
4. *$\mathrm{snf}(\phi_1 \circ \phi_2) = \mathrm{snf}(\phi_1) \circ \mathrm{snf}(\phi_2)$, where $\circ \in \{\wedge, \vee\}$*
5. *$\mathrm{snf}(\circ\phi) = \circ\phi$ where $\circ \in \{\mathsf{X}, \mathsf{Y}, \widetilde{\mathsf{X}}, \mathsf{Z}\}$;*
6. *$\mathrm{snf}(\phi_1 \mathsf{U} \phi_2) = \mathrm{snf}(\phi_2) \vee (\mathrm{snf}(\phi_1) \wedge \mathsf{X}(\phi_1 \mathsf{U} \phi_2))$;*
7. *$\mathrm{snf}(\phi_1 \mathsf{R} \phi_2) = \mathrm{snf}(\phi_2) \wedge (\mathrm{snf}(\phi_1) \vee \widetilde{\mathsf{X}}(\phi_1 \mathsf{R} \phi_2))$;*
8. *$\mathrm{snf}(\phi_1 \mathsf{S} \phi_2) = \mathrm{snf}(\phi_2) \vee (\mathrm{snf}(\phi_1) \wedge \mathsf{Y}(\phi_1 \mathsf{S} \phi_2))$;*
9. *$\mathrm{snf}(\phi_1 \mathsf{T} \phi_2) = \mathrm{snf}(\phi_2) \wedge (\mathrm{snf}(\phi_1) \vee \mathsf{Z}(\phi_1 \mathsf{T} \phi_2))$;*

Any temporal operator in $\mathrm{snf}(\phi)$ appears only directly below one among $\mathsf{X}, \mathsf{Y}, \widetilde{\mathsf{X}}, \mathsf{Z}$. Intuitively, $\mathrm{snf}(\phi)$ separates what the formula says about the current state, from what it says about the next and previous ones. The following defines a universe of formulas that are important for the satisfaction of a given sentence.

Definition 10 (Closure)**.** *The* closure *of an* FOLTL *sentence ϕ is the set $\mathcal{C}(\phi)$ defined as follows:*

1. *$\mathsf{X}\phi \in \mathcal{C}(\phi)$;*
2. *$\psi \in \mathcal{C}(\phi)$ for any subformula ψ of ϕ (including itself);*
3. *for any $\phi_1 \mathsf{U} \phi_2 \in \mathcal{C}(\phi)$, $\phi_1 \mathsf{R} \phi_2 \in \mathcal{C}(\phi)$, $\phi_1 \mathsf{S} \phi_2 \in \mathcal{C}(\phi)$, or $\phi_1 \mathsf{T} \phi_2 \in \mathcal{C}(\phi)$, we have $\mathsf{X}(\phi_1 \mathsf{U} \phi_2) \in \mathcal{C}(\phi)$, $\widetilde{\mathsf{X}}(\phi_1 \mathsf{R} \phi_2) \in \mathcal{C}(\phi)$, $\mathsf{Y}(\phi_1 \mathsf{S} \phi_2) \in \mathcal{C}(\phi)$, or $\mathsf{Z}(\phi_1 \mathsf{T} \phi_2) \in \mathcal{C}(\phi)$, respectively.*

Now, we define some symbols, called *surrogates*, that take the place of temporal formulas in the encoding below.

$$\mathrm{XS} = \{\mathsf{xs}_\psi \mid \mathsf{X}\psi \in \mathcal{C}(\phi)\} \quad \widetilde{\mathrm{X}}\mathrm{S} = \{\mathsf{ws}_\psi \mid \widetilde{\mathsf{X}}\psi \in \mathcal{C}(\phi)\}$$
$$\mathrm{YS} = \{\mathsf{ys}_\psi \mid \mathsf{Y}\psi \in \mathcal{C}(\phi)\} \quad \mathrm{ZS} = \{\mathsf{zs}_\psi \mid \mathsf{Z}\psi \in \mathcal{C}(\phi)\}$$

where the xs_ψ, ws_ψ, ys_ψ, zs_ψ are predicates of the arity n corresponding to the number of free first-order variables in ψ. For every $\psi \in \mathcal{C}(\phi)$, we denote with $\mathrm{snf}_S(\psi)$ the formula obtained from $\mathrm{snf}(\psi)$ by replacing each $\mathsf{X}\theta$, $\widetilde{\mathsf{X}}\theta$, $\mathsf{Y}\theta$, or $\mathsf{Z}\theta$ by their surrogates xs_θ, ws_θ, ys_θ, or zs_θ, respectively. The formula $\mathrm{snf}'_S(\psi)$ is defined similarly but using the primed xs'_θ, ws'_θ, ys'_θ, and zs'_θ. Then we can finally proceed.

Encoding 1 (From FOLTL to automata)**.** *Let ϕ be an* FOLTL *sentence. We define $\mathcal{A}(\phi) = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ as follows:*

1. *the state signature is $\Gamma = \mathrm{XS} \cup \widetilde{\mathrm{X}}\mathrm{S} \cup \mathrm{YS} \cup \mathrm{ZS}$;*
2. *the initial and final conditions are:*

$$\phi_0 \coloneqq \mathsf{xs}_\phi \wedge \bigwedge_{\mathsf{zs}_\psi \in \mathrm{ZS}} \forall \overline{x}.\mathsf{zs}_\psi(\overline{x}) \wedge \bigwedge_{\mathsf{ys}_\psi \in \mathrm{YS}} \forall \overline{x}.\neg \mathsf{ys}_\psi(\overline{x})$$

$$\phi_F \coloneqq \bigwedge_{\mathsf{ws}_\psi \in \widetilde{\mathrm{X}}\mathrm{S}} \forall \overline{x}.\mathsf{ws}_\psi(\overline{x}) \wedge \bigwedge_{\mathsf{xs}_\psi \in \mathrm{XS}} \forall \overline{x}.\neg \mathsf{xs}_\psi(\overline{x})$$

3. *the transition relation is:*

$$\phi_T \coloneqq \bigwedge_{s_\psi \in \mathrm{XS} \cup \widetilde{\mathrm{X}}\mathrm{S}} \forall \overline{x}.[s_\psi(\overline{x}) \leftrightarrow \mathrm{snf}'_S(\psi(\overline{x}))] \quad (1)$$

$$\wedge \bigwedge_{s_\psi \in \mathrm{YS} \cup \mathrm{ZS}} \forall \overline{x}.[s'_\psi(\overline{x}) \leftrightarrow \mathrm{snf}_S(\psi(\overline{x}))] \quad (2)$$

Intuitively, the initial condition states that, in the first component of the word, ϕ must hold and all formulas of type $\mathsf{Y}\psi$ (resp., $\mathsf{Z}\psi$) in the closure are false (resp., true), following FOLTL semantics. Then, the transition relation ensures that the surrogates of all the future and past formulas behave following the semantics of such formulas. The final condition ensures we are at the end of a finite word, by checking whether all formulas of type $\mathsf{X}\psi$ (resp., $\widetilde{\mathsf{X}}\psi$) in the closure are false (resp., true). We can state and prove the correctness of the above construction.

Theorem 6. *For any* FOLTL *sentence ϕ and any theory $\mathcal{T}$, we have $\mathcal{L}_\mathcal{T}(\mathcal{A}(\phi)) = \mathcal{L}_\mathcal{T}(\phi)$.*

5 Deterministic Automata From Pure-Past FOLTL

In this section, we consider the fragment of FOLTL in which temporal operators are restricted to talk about the past. We show how to go *directly* from formulas of this fragment to *deterministic* first-order automata and we discuss the importance of such construction in practical settings.

The pure-past fragment of FOLTL

A *pure-past* FOLTL formula is a formula ϕ which does *not* use future temporal operators (X, $\widetilde{\mathsf{X}}$, U, R). Pure-past FOLTL is denoted as FOpLTL. Given that only past operators are available, it is natural to interpret a FOpLTL formula at the *end* of a word. Hence given a $\mathcal{T}$-word $\overline{\sigma}$ and a FOpLTL formula ϕ, we say that $\overline{\sigma} = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle$ satisfies ϕ, denoted $\overline{\sigma} \models \phi$, iff $\overline{\sigma}, n-1 \models \phi$ according to the semantics of FOLTL defined in Section 2. $\mathcal{L}_\mathcal{T}(\phi)$ is the set of $\mathcal{T}$-words satisfying ϕ. The construction of Encoding 1 can be tailored to FOpLTL formulas.

Encoding 2 (Automaton of a FOpLTL sentence)**.** *Let ϕ be a* FOpLTL *sentence. We define $\mathcal{A}(\phi) = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ similarly to Encoding 1 with the following changes:*

1. *$\mathcal{C}(\phi)$ is defined as in Definition 10 but Item 1 mentions $\mathsf{Y}\phi$ instead of $\mathsf{X}\phi$;*
2. *the initial state condition is defined as follows:*

$$\phi_0 \coloneqq \bigwedge_{\mathsf{zs}_\psi \in \mathrm{ZS}} \forall \overline{x}\, \mathsf{zs}_\psi(\overline{x}) \bigwedge_{\mathsf{ys}_\psi \in \mathrm{YS}} \forall \overline{x}\, \neg \mathsf{ys}_\psi(\overline{x})$$

3. *the final state condition is defined as $\phi_F \coloneqq \mathsf{ys}_\phi$.*

Proving that this revised encoding is correct is a matter of minor modifications to the proof of Theorem 6. However, one may wonder what we gained with this construction. The crucial feature is that the automaton resulting from this encoding of a pure-past sentence is *deterministic*.

Theorem 7. *Given a* FOpLTL *sentence ϕ, $\mathcal{A}(\phi)$ is* complete *and* deterministic.

The relevance of the pure-past fragment

Pure-past fragments of temporal logics (Lichtenstein, Pnueli, and Zuck 1985) are gaining renewed interest for their practical use in formal verification, such as in *reactive synthesis* and *monitoring*.

Reactive synthesis (Pnueli and Rosner 1989) involves creating a system that satisfies a temporal specification. For a temporal formula ϕ with *controllable* ($\mathcal{C}$) and *uncontrollable* ($\mathcal{U}$) variables, it seeks to synthesize a controller that ensures ϕ is satisfied regardless of $\mathcal{U}$. For LTL, this involves generating a *deterministic* automaton $\mathcal{A}_\phi$ and solving fixpoint computations. Generating such automata is complex and typically requires *doubly* exponential time. However, for pure-past LTL formulas, deterministic finite automata can be built directly in *singly* exponential time, making the problem only EXPTIME-complete (Artale et al. 2023). Symbolic techniques are also effective here, allowing efficient representation of these automata.

Monitoring (Leucker and Schallhart 2009) is a runtime verification technique that detects violations of temporal properties during system execution. To ensure irrevocable verdicts, it relies on deterministic automata, which are challenging to produce from temporal specifications (Kupferman and Vardi 1999). The use of pure-past FOLTL is seen as promising for monitoring in infinite-state settings, with deterministic automata being essential to this approach.

6 Checking Emptiness

Given Theorem 6, we know that the emptiness problem for first-order automata is highly undecidable, since so is FOLTL satisfiability, in general (Gabbay et al. 2003). However, we can show how to at least recover semi-decidability in some cases, setting the stage for future algorithmic developments. To be more specific, what can be semi-decided, in some cases, is the *non-emptiness* problem of first-order automata, which translates to the *satisfiability* of a certain class of FOLTL formulas.

For $k \geq 0$ and any sentence ψ over a signature Σ and a Σ-theory $\mathcal{T}$, we denote as ψ^k, Σ^k, $\mathcal{T}^k$, respectively, the objects obtained by *renaming* any *non-rigid* symbol s by s^k, and any *primed non-rigid* symbol s' by s^{k+1}. Now, given an automaton $\mathcal{A} = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$, we define, for some $k \geq 0$, the formulas $[\![\mathcal{A}]\!]_k \equiv \phi_0^0 \wedge \bigwedge_{i=0}^{k-1} \phi_T^i$ and $[\![\mathcal{A}]\!]_k^F \equiv [\![\mathcal{A}]\!]_k \wedge \phi_F^k$. Intuitively, $[\![\mathcal{A}]\!]_k^F$ represents the *accepted* runs of $\mathcal{A}$ of length at most $k+1$.

We can use $[\![\mathcal{A}]\!]_k^F$ to define an iterative procedure *à la* Bounded Model Checking (Biere et al. 2009), shown in Algorithm 1, where at each $k > 0$ we test if $[\![\mathcal{A}]\!]_k^F$ is satisfiable

Algorithm 1: Non-emptiness semi-decision

```
1: procedure NonEmpty($\mathcal{A}$)
2:     for $k \leftarrow 0 \ldots +\infty$ do
3:         if $[\![\mathcal{A}]\!]_k^F$ is $\mathcal{T}^{0\ldots k}$-satisfiable then
4:             return not empty
5:         end if
6:     end for
7: end procedure
```

and we increment k if it is not. Note that $[\![\mathcal{A}]\!]_k^F$ is a formula over $\Sigma^{0\ldots k-1} = \bigcup_{i=0}^{k-1} \Sigma^i$ and $\Gamma^{0\ldots k} = \bigcup_{i=0}^{k} \Gamma^i$, to be interpreted over $\mathcal{T}^{0\ldots k} = \bigcup_{i=0}^{k} \mathcal{T}^k$. One can prove the following.

Proposition 3. *$[\![\mathcal{A}]\!]_k^F$ is $\mathcal{T}^{0\ldots k}$-satisfiable iff $\mathcal{L}_{\mathcal{T}}(\mathcal{A}) \neq \varnothing$.*

Therefore, we need to test $\mathcal{T}^{0\ldots k}$-satisfiability of $[\![\mathcal{A}]\!]_k^F$, and we need this test to be decidable. This is, of course, not always possible, but it can be depending on the shape of $\mathcal{A}$ and the considered theory. Precisely, we have the following.

Theorem 8. *Algorithm 1 is a semi-decision procedure for first-order automata non-emptiness if satisfiability of ϕ_0 and of ϕ_F and $\mathcal{T}$-satisfiability of ϕ_T are decidable.*

Theorem 8 might seem a rather weak result in practice, but it is worth to note that it is in fact a more general statement of what is already implemented in the BLACK temporal reasoning framework (Geatti, Gigante, and Montanari 2019; Geatti et al. 2024; Geatti, Gigante, and Montanari 2021) when dealing with $\mathsf{LTL_f}$ *modulo theories* ($\mathsf{LTL_f^{MT}}$), a restricted fragment of FOLTL with rigid predicates and non-rigid constants (Geatti, Gianola, and Gigante 2022; Geatti et al. 2023). By leveraging modern SMT solvers (Barrett et al. 2021), BLACK manages to get promising performance on satisfiable instances.

7 Decidable Cases

First-order automata are a really general formalism and this generality is paid with the undecidability of their emptiness. In this section, we show some cases when decidability can be recovered by restricting what can appear in the *state signature* of the automata, and relate these results to some models from the literature that can be captured by first-order automata at different levels of restriction.

Definition 11 (State signature restrictions)**.** *An automaton $\mathcal{A}$ with state signature Γ is said to be:*

- finite-control*, if Γ consists of propositions;*
- data-control*, if Γ consists of propositions or constants;*
- monadic*, if Γ is a relational signature with only* monadic *(*i.e.*, unary) predicates.*

Finite-control automata. Finite-control automata are basically *finite-state* machines, but can read first-order alphabets. The finite number of different control states makes it easier to handle the emptiness problem.

Theorem 9. *Under the same conditions of Theorem 8, emptiness of finite-control automata is decidable.*

Finite-control automata may seem limited but they are still quite expressive, given the ability of treating infinite alphabets. Indeed, much literature is devoted to *symbolic finite automata* (s-FA) (Veanes et al. 2012; D'Antoni and Veanes 2014; D'Antoni, Kincaid, and Wang 2018; D'Antoni and Veanes 2017; Tamm and Veanes 2018), which are finite automata able to read large or infinite alphabets defined by effective Boolean algebras. The state space of s-FAs is finite and emptiness is decidable for them as well. Indeed, finite-control first-order automata can easily encode s-FAs (see (Geatti, Gianola, and Gigante 2024)). From a logical perspective, following Encoding 1, we can see that finite-control automata can encode FOLTL formulas where temporal operators are only applied to *sentences* (*i.e.*, closed formulas).

Data-control automata. In *data-control* automata the state space returns to be infinite, but loosely structured. Intuitively, the constants in Γ act as storage registers for single pieces of data coming from the respective domains. It turns out this is sufficient to capture the $\mathsf{LTL_f^{MT}}$ logic (Geatti, Gianola, and Gigante 2022; Geatti et al. 2023) mentioned in Section 6. This logic restricts FOLTL as follows: a) only constants can be non-rigid; b) quantifiers cannot be applied to temporal formulas. At the same time, $\mathsf{LTL_f^{MT}}$ *extends* FOLTL with *lookahead* term constructors $\bigcirc c$ and $\circledsim c$, for some constant c, which evaluate to the value of c at the next time step (strongly or weakly, respectively, since we are on finite words). For example, a predicate such as $p(\bigcirc c)$ means that p holds now on the value that c will have at the next state (which is required to exist). FOLTL can express lookaheads as follows:

$$p(\bigcirc c) \equiv \exists x.(p(x) \wedge \mathsf{X}(c = x))$$

By Encoding 1, the automaton encoding this formula has a predicate in Γ acting as the surrogate for the formula $\mathsf{X}(c = x)$. However, the predicate always holds only for a single value, the one equal to c at the next state. Therefore, one can see that these predicates can be replaced by constants, and thus $\mathsf{LTL_f^{MT}}$ formulas can be captured by data-control automata. This is interesting since it opens the way to transfering decidability results from some known fragments of $\mathsf{LTL_f^{MT}}$ (Geatti et al. 2023) to suitable subclasses of data-control automata. Such fragments are based on a notion of *finite summary*, that intuitively requires the history of the word to be summarisable at any time in a finite number of ways. A similar criterion is used in works from the *process modeling* community (Felli, Montali, and Winkler 2022; Gianola, Montali, and Winkler 2023, 2024) to obtain decidability of the emptiness problem for a class of state machines called *data-aware processes modulo theories* (DMT), whose control states are described by the values of a finite set of variables over the theory's domain. Data-control automata can capture DMTs quite easily, with the constants in Γ doing the job of such state variables (see (Geatti, Gianola, and Gigante 2024) for details). Therefore, we conjecture that a decidability criterion similar to DMT's finite summary can be defined for data-control first-order automata as well.

It is also worth pointing out that data-control automata are suitable to formalise the semantics of the MoXI modeling language (Johannsen et al. 2024; Rozier et al. 2024). MoXI is a general intermediate language for the modeling of finite- and infinite-state model checking problems that is recently being developed by the community in the open as a central resource for research and experimentation. The language supports the description of infinite-state systems through first-order models, but with the evolution over time only of data variables. Although showing all the details of this formalisation is outside of the scope of this paper, one can see that models described through MoXI are covered by data-control automata.

Monadic automata. From Encoding 1, one can see that the whole FOLTL can be encoded using first-order automata whose state signature Γ is purely relational. More precisely, the arity of the predicates in Γ depends on the number of *free variables* of temporal subformulas. If we start from *monodic* sentences, *i.e.*, those where temporal subformulas have at most *one* free variable, we obtain a state signature with only *monadic* predicates. This connection is interesting because of the following classic result.

Proposition 4 (Decidability of monodic FOLTL (Gabbay et al. 2003))**.** *Let Σ be a* relational *signature with* rigid constants*, $\mathcal{F}$ a class of first-order Σ-sentences* without equality*, and $\mathcal{T}$ a Σ-theory, such that $\mathcal{T}$-satisfiability of $\mathcal{F}$-sentences is decidable. Then, $\mathcal{T}$-satisfiability of* monodic FOLTL *$\mathcal{F}$-sentences is decidable.*

One may wonder whether Proposition 4 can be showed via monadic automata, under the same assumptions. It turns out it can and, more precisely, we can show that any monadic automaton can be translated into an equivalent finite-control one, then using Theorem 9 to get decidability.

Behind this encoding is the observation that monadic predicates cannot express any kind of relationship between different elements of the domains. The only aspect one can model is the existence or not of elements satisfying certain combinations of predicates, but elements which satisfy the exact same predicates are indistinguishable. The domains can therefore be partitioned into a finite number of classes, and this partition can be described by propositions. The following result, with Theorem 9, directly proves Proposition 4.

Theorem 10. *Let $\mathcal{A} = \langle \Sigma, \Gamma, \phi_0, \phi_T, \phi_F \rangle$ be a* monadic *automaton where ϕ_0, ϕ_T, and ϕ_F do not use equality. Then, there is a* finite-control *automaton $\mathcal{A}'$ equivalent to $\mathcal{A}$.*

8 Discussion, Related Work, and Future Directions

We introduced *first-order automata*, an automaton model capable of capturing FOLTL on finite words. We studied the corresponding notion of $\mathcal{T}$-regular languages, and how to deal with the (non-)emptiness problem in some cases.

Of course, other models capable of describing infinite-state behaviors have been proposed before. Many have been introduced in the last decades, including timed automata (Alur and Dill 1994), hybrid automata (Alur et al. 1992), recursive state machines (Alur et al. 2005; Benerecetti, Minopoli, and Peron 2010), visibly pushdown languages (Alur and Madhusudan 2004), operator-precedence automata (Droste et al. 2017), FIFO machines (Brand and Zafiropulo 1983), counter machines (Wojna 1999), Petri nets (Murata 1989; Jensen and Kristensen 2009), data-aware processes (Deutsch, Li, and Vianu 2019; Calvanese, De Giacomo, and Montali 2013; Ghilardi et al. 2023; Gianola 2023), many automata over infinite alphabets (Segoufin 2006), register automata (Kaminski and Francez 1994), and many more. However, most of them have been designed to address specific problems, in specific kinds of scenarios, with good computational properties, so they cannot be general enough to capture FOLTL which is highly undecidable.

Symbolic finite automata (s-FA) (D'Antoni, Kincaid, and Wang 2018; D'Antoni and Veanes 2014, 2017), which we mentioned in Section 7, share some similarity with ours, although they are less expressive because of their decidability. *Alternating symbolic automata* (Iosif and Xu 2019) are more expressive than s-FAs, closed under complementation and with a semi-decidable emptiness problem, but are limited to words that, in our setting, would be made of a signature Σ with only propositions and non-rigid constants.

Our automaton model is a valuable tool for reasoning about FOLTL, as evidenced by our proof of Proposition 4. Although the original proof is more general (it works also on $(\mathbb{Q}, <)$, $(\mathbb{R}, <)$, and any class of infinite or finite first-order definable linear orders), ours is much more direct in the particular case of finite discrete linear orders, which are those naturally handled by automata. Automata are also crucial for FOLTL model checking, that can now be reduced to the emptiness problem. How to deal with this problem efficiently in practice is a different issue. Building on top of existing SMT technology (Barrett et al. 2021), a promising path would be to lift known techniques, such as CEGAR (Clarke et al. 2000), interpolation (McMillan 2018), and property-directed reachability (Bradley 2012). Constrained Horn clauses (Gurfinkel and Bjørner 2019) are also promising for *quantifier-free* first-order automata, while for quantified ones, quantifier elimination (Calvanese et al. 2022, 2021), also of second-order predicates (Gabbay, Schmidt, and Szalas 2008), has to be used.

Beyond verification, reactive synthesis (Pnueli and Rosner 1989) is the most ambitious goal. As we discussed extensively in Section 5, *deterministic* automata are key for solving this problem. Most promisingly, these can be obtained either from pure-past FOLTL sentences, as explained in Section 5. Recent results (Rodríguez and Sánchez 2023) addressed reactive synthesis for simple fragments of $\mathsf{LTL}_\mathsf{f}^\mathsf{MT}$ (hence of FOLTL) with encouraging results. For more general FOLTL specifications, however, more work is needed.

An extension of first-order automata to *infinite words* is the natural next step. One can easily define Büchi, Streett, or Rabin first-order ω-automata by replacing ϕ_F with suitable sentences representing the sets of states used in such acceptance conditions. Then, FOLTL on infinite words is easily encoded into such Streett first-order ω-automata by extending Encoding 1, although we could not include it in this paper for space reasons. However, first-order ω-automata would pose even more challenges: all the theory about ω-regular languages has to be rebuilt from scratch, and dealing with them algorithmically is going to be even harder.

Acknowledgements

Luca Geatti acknowledges the support from the Interconnected Nord-Est Innovation Ecosystem (iNEST), which received funding from the European Union NextGenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.5 – D.D. 1058 23/06/2022, ECS00000043). Alessandro Gianola was partially supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under projects UIDB/50021/2020 (DOI:10.54499/UIDB/50021/2020).

References

Alur, R.; Benedikt, M.; Etessami, K.; Godefroid, P.; Reps, T. W.; and Yannakakis, M. 2005. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4): 786–818.

Alur, R.; Courcoubetis, C.; Henzinger, T. A.; and Ho, P. 1992. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, 209–229. Springer.

Alur, R.; and Dill, D. L. 1994. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2): 183–235.

Alur, R.; and Madhusudan, P. 2004. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, 202–211. ACM.

Artale, A.; Geatti, L.; Gigante, N.; Mazzullo, A.; and Montanari, A. 2023. Complexity of Safety and coSafety Fragments of Linear Temporal Logic. In *Proceedings of AAAI 2023*, 6236–6244. AAAI Press.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2021. Satisfiability Modulo Theories. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 1267–1329. IOS Press.

Benerecetti, M.; Minopoli, S.; and Peron, A. 2010. Analysis of Timed Recursive State Machines. In *Proceedings of TIME 2010*, 61–68. IEEE Computer Society.

Biere, A.; Cimatti, A.; Clarke, E. M.; Strichman, O.; and Zhu, Y. 2009. Bounded model checking. *Handbook of satisfiability*, 185(99): 457–481.

Bradley, A. R. 2012. Understanding IC3. In *Procedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, volume 7317 of *Lecture Notes in Computer Science*, 1–14. Springer.

Brand, D.; and Zafiropulo, P. 1983. On Communicating Finite-State Machines. *J. ACM*, 30(2): 323–342.

Büchi, J. R. 1962. On a decision method in restricted second order arithmetic. In Nagel, E.; Suppes, P.; and Tarski, A., eds., *Proceedings of the 1960 International Congress on Logic, methodology and philosophy of science*, 1–11. Stanford University Press.

Calvanese, D.; De Giacomo, G.; and Montali, M. 2013. Foundations of data-aware process analysis: a database theory perspective. In *Proceedings of PODS 2013*, 1–12. ACM.

Calvanese, D.; Ghilardi, S.; Gianola, A.; Montali, M.; and Rivkin, A. 2021. Model Completeness, Uniform Interpolants and Superposition Calculus. *J. Autom. Reason.*, 65(7): 941–969.

Calvanese, D.; Ghilardi, S.; Gianola, A.; Montali, M.; and Rivkin, A. 2022. Combination of Uniform Interpolants via Beth Definability. *J. Automat. Reason.*, 66(3): 409–435.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-Guided Abstraction Refinement. In *Proceedings of 12th CAV*, volume 1855 of *Lecture Notes in Computer Science*, 154–169. Springer.

Clarke, E. M.; Grumberg, O.; Kroening, D.; Peled, D. A.; and Veith, H. 2018. *Model checking, 2nd Edition*. MIT Press. ISBN 978-0-262-03883-6.

D'Antoni, L.; Kincaid, Z.; and Wang, F. 2018. A Symbolic Decision Procedure for Symbolic Alternating Finite Automata. In *Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics*, volume 341 of *Electronic Notes in Theoretical Computer Science*, 79–99. Elsevier.

D'Antoni, L.; and Veanes, M. 2014. Minimization of symbolic automata. In *Proceedings of POPL 2014*, 541–554. ACM.

D'Antoni, L.; and Veanes, M. 2017. The Power of Symbolic Automata and Transducers. In *Proceedings of the 29th CAV*, volume 10426 of *Lecture Notes in Computer Science*, 47–67. Springer.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proceedings of 23rd IJCAI*, 854–860.

Deutsch, A.; Li, Y.; and Vianu, V. 2019. Verification of Hierarchical Artifact Systems. *ACM Trans. Database Syst.*, 44(3): 12:1–12:68.

Droste, M.; Dück, S.; Mandrioli, D.; and Pradella, M. 2017. Weighted Operator Precedence Languages. In *Procedings of the 42nd International Symposium on Mathematical Foundations of Computer Science*, volume 83 of *LIPIcs*, 31:1–31:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Felli, P.; Montali, M.; and Winkler, S. 2022. Linear-Time Verification of Data-Aware Dynamic Systems with Arithmetic. In *Proceedings of AAAI 2022*, 5642–5650. AAAI Press.

Gabbay, D.; Kurucz, A.; Wolter, F.; and Zakharyaschev, M. 2003. Fragments of first-order temporal logics. In *Many-Dimensional Modal Logics*, volume 148 of *Studies in Logic and the Foundations of Mathematics*, 465–545. Elsevier.

Gabbay, D. M.; Schmidt, R. A.; and Szalas, A. 2008. *Second-Order Quantifier Elimination - Foundations, Computational Aspects and Applications*, volume 12 of *Studies in logic : Mathematical logic and foundations*. College Publications. ISBN 978-1-904987-56-7.

Geatti, L.; Gianola, A.; and Gigante, N. 2022. Linear Temporal Logic Modulo Theories over Finite Traces. In *Proceedings of the 31st IJCAI*, 2641–2647.

Geatti, L.; Gianola, A.; and Gigante, N. 2024. A General Automata Model for First-Order Temporal Logics (Extended Version). *CoRR*, abs/2405.20057.

Geatti, L.; Gianola, A.; Gigante, N.; and Winkler, S. 2023. Decidable Fragments of LTL$_f$ Modulo Theories. In *Proceedings of the 26th ECAI*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, 811–818. IOS Press.

Geatti, L.; Gigante, N.; and Montanari, A. 2019. A SAT-Based Encoding of the One-Pass and Tree-Shaped Tableau System for LTL. In *Proceedings of the 28th TABLEAUX*, volume 11714 of *LNCS*, 3–20.

Geatti, L.; Gigante, N.; and Montanari, A. 2021. BLACK: A Fast, Flexible and Reliable LTL Satisfiability Checker. In *Proceedings of the 3rd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis*, volume 2987 of *CEUR Workshop Proceedings*, 7–12. CEUR-WS.org.

Geatti, L.; Gigante, N.; Montanari, A.; and Venturato, G. 2024. SAT Meets Tableaux for Linear Temporal Logic Satisfiability. *J. Autom. Reason.*, 68(2): 6.

Ghilardi, S.; Gianola, A.; Montali, M.; and Rivkin, A. 2023. Safety Verification and Universal Invariants for Relational Action Bases. In *Proceedings of the 32nd IJCAI*, 3248–3257. ijcai.org.

Gianola, A. 2023. *Verification of Data-Aware Processes via Satisfiability Modulo Theories*, volume 470 of *Lecture Notes in Business Information Processing*. Springer.

Gianola, A.; Montali, M.; and Winkler, S. 2023. Linear-Time Verification of Data-Aware Processes Modulo Theories via Covers and Automata (Extended Version). *CoRR*, abs/2310.12180.

Gianola, A.; Montali, M.; and Winkler, S. 2024. Linear-Time Verification of Data-Aware Processes Modulo Theories via Covers and Automata. In *Proceedings of AAAI 2024*, 10525–10534. AAAI Press.

Gurfinkel, A.; and Bjørner, N. S. 2019. The Science, Art, and Magic of Constrained Horn Clauses. In *Proceedings of the 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 6–10. IEEE.

Iosif, R.; and Xu, X. 2019. Alternating Automata Modulo First Order Theories. In *Proceedings of the 31st CAV*, volume 11562 of *Lecture Notes in Computer Science*, 43–63. Springer.

Jensen, K.; and Kristensen, L. M. 2009. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer.

Johannsen, C.; Nukala, K.; Dureja, R.; Irfan, A.; Shankar, N.; Tinelli, C.; Vardi, M. Y.; and Rozier, K. Y. 2024. The MoXI Model Exchange Tool Suite. In *Proceedings of the 36th CAV*, volume 14681 of *Lecture Notes in Computer Science*, 203–218. Springer.

Kaminski, M.; and Francez, N. 1994. Finite-Memory Automata. *Theor. Comput. Sci.*, 134(2): 329–363.

Kontchakov, R.; Lutz, C.; Wolter, F.; and Zakharyaschev, M. 2004. Temporalising Tableaux. *Stud Logica*, 76(1): 91–134.

Kupferman, O.; and Vardi, M. Y. 1999. Model Checking of Safety Properties. In *Proceedings of the 11th CAV*, volume 1633 of *Lecture Notes in Computer Science*, 172–183. Springer.

Leucker, M.; and Schallhart, C. 2009. A brief account of runtime verification. *J. Log. Algebraic Methods Program.*, 78(5): 293–303.

Lichtenstein, O.; Pnueli, A.; and Zuck, L. 1985. The glory of the past. In *Workshop on Logic of Programs*, 196–218. Springer.

Loeckx, J.; and Sieber, K. 1987. *The Foundations of Program Verification, 2nd ed.* Wiley-Teubner. ISBN 3-519-12101-8.

McMillan, K. L. 2018. Interpolation and Model Checking. In Clarke, E. M.; Henzinger, T. A.; Veith, H.; and Bloem, R., eds., *Handbook of Model Checking*, 421–446. Springer.

Murata, T. 1989. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4): 541–580.

Pnueli, A. 1977. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science*, 46–57. IEEE Computer Society.

Pnueli, A.; and Rosner, R. 1989. On the Synthesis of an Asynchronous Reactive Module. In *16th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, 652–671. Springer.

Rodríguez, A.; and Sánchez, C. 2023. Boolean Abstractions for Realizability Modulo Theories. In *Proceedings of the 35th CAV*, volume 13966 of *Lecture Notes in Computer Science*, 305–328. Springer.

Rozier, K. Y.; Dureja, R.; Irfan, A.; Johannsen, C.; Nukala, K.; Shankar, N.; Tinelli, C.; and Vardi, M. Y. 2024. Moxi: An intermediate language for symbolic model checking. In *Proceedings of the 30th International Symposium on Model Checking Software*.

Segoufin, L. 2006. Automata and Logics for Words and Trees over an Infinite Alphabet. In *Proceedings of the 20th International Workshop on Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, 41–57. Springer.

Tamm, H.; and Veanes, M. 2018. Theoretical Aspects of Symbolic Automata. In *44th International Conference on Theory and Practice of Computer Science*, volume 10706 of *Lecture Notes in Computer Science*, 428–441. Springer.

Vardi, M. Y. 1991. Verification of concurrent programs: The automata-theoretic framework. *Annals of Pure and Applied Logic*, 51(1-2): 79–98.

Vardi, M. Y. 2005. An automata-theoretic approach to linear temporal logic. *Logics for concurrency: structure versus automata*, 238–266.

Veanes, M.; Hooimeijer, P.; Livshits, B.; Molnar, D.; and Bjørner, N. S. 2012. Symbolic finite state transducers: algorithms and applications. In *Proceedings of POPL 2012*, 137–150. ACM.

Wojna, A. 1999. Counter Machines. *Inf. Process. Lett.*, 71(5-6): 193–197.