Information and Computation 251 (2016) 104–139

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

On temporal logics with data variable quantifications: Decidability and complexity[☆]

Fu Song[a,b], Zhilin Wu[c,d,*]

[a] *School of Information Science and Technology, ShanghaiTech University, Shanghai, PR China*
[b] *Shanghai Key Laboratory of Trustworthy Computing, National Trusted Embedded Software Engineering Technology Research Center, East China Normal University, PR China*
[c] *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, PR China*
[d] *LIAFA, Université Paris Diderot, France*

ARTICLE INFO

Article history:
Received 5 August 2015
Received in revised form 13 June 2016
Available online 16 August 2016

Keywords:
Temporal logics
Data variable quantifications
Satisfiability
Model checking
Decidability and complexity
Alternating register automata
Data automata

ABSTRACT

Although data values are available in almost every computer system, reasoning about them is a challenging task due to the huge data size or even infinite data domains. Temporal logics are the well-known specification formalisms for reactive and concurrent systems. Various extensions of temporal logics have been proposed to reason about data values, mostly in the last decade. Among them, one natural idea is to extend temporal logics with variable quantifications ranging over an infinite data domain. Grumberg, Kupferman and Sheinvald initiated the research on this topic recently and obtained several interesting results. However, there is still a lack of systematic investigation on the theoretical aspects of the variable extensions of temporal logics. Our goal in this paper is to fill this gap. Around this goal, we make the following choices: 1. We consider the variable extensions of two widely used temporal logics, Linear Temporal Logic (LTL) and Computation Tree Logic (CTL), and allow arbitrary nestings of variable quantifications with Boolean and temporal operators (the resulting logics are called respectively variable-LTL, in brief VLTL, and variable-CTL, in brief VCTL). 2. We investigate the decidability and complexity of both the satisfiability and model checking problems, over both finite and infinite words (trees). In particular, we obtain the following results: Existential variable quantifiers or one single universal quantifier in the beginning already entail the undecidability of the satisfiability problems of both VLTL and VCTL, over both finite and infinite words (trees); if only existential path quantifiers are used in VCTL, then the satisfiability problem (over finite trees) is decidable, no matter which variable quantifiers are available; for VLTL formulae with one single universal variable quantifier in the beginning, if the occurrences of the non-parameterized atomic propositions are guarded by the positive occurrences of the quantified variables, then its satisfiability problem becomes decidable, over both finite and infinite words; based on these results of the satisfiability problem, we deduce the (un)decidability results of the model checking problem.

[☆] A preliminary version of this paper appeared in FSTTCS 2014 (the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science), under the title "Extending temporal logics with data variable quantifications".

[*] Corresponding author at: P.O. Box 8718, #4 South 4th Street, Zhongguancun, Haidian district, 100190, Beijing, PR China.
E-mail address: wuzl@ios.ac.cn (Z. Wu).

http://dx.doi.org/10.1016/j.ic.2016.08.002
0890-5401/© 2016 Elsevier Inc. All rights reserved.

1. Introduction

Context. Data values are ubiquitous in computer systems. To see just the tip of the iceberg, we have the following scenarios: data variables in sequential programs, process identifiers in concurrent parameterized systems where an unbounded number of processes interact with each other, records in relational databases, attributes of elements in XML documents or nodes in graph databases. On the other hand, reasoning about data values is a very challenging task. Either their sizes are huge, e.g. one single 4-byte integer variable in C programs may take values from $-2,147,483,648$ to $2,147,483,647$, or they are even from an infinite domain, e.g. process identifiers in parameterized systems.

Temporal logics are the formalisms widely used to specify the behaviors of concurrent, reactive as well as sequential systems. Linear Temporal Logic (LTL) [1] and Computation Tree Logic (CTL) [2] are the two most widely used temporal logics. Although temporal logics were originally targeted to specify the behaviors of finite state systems, various extensions of temporal logics have been proposed to deal with the infinite data values in computer systems (mostly in the last decade).

- First-order temporal logics, that is, first-order logic over relational structures extended with temporal operators, are a natural formalism to specify and reason about infinite data values in computer systems. Although most of the work about temporal logics focuses on the propositional ones, first-order temporal logics were investigated in fact at almost the same time as their propositional counterparts (see e.g. [3]). Initially, the investigations were around the axiomatization issues [4–6]. Later on, Hodkinson et al. started investigating the decidability and complexity of the satisfiability problem [7,8].
- Vianu and his coauthors used the first-order extensions of LTL to specify and reason about the behaviors of database-driven systems [9,10].
- On the other hand, Demri and Lazic extended LTL with freeze quantifiers which can store data values into registers and compare the data values with those stored in the registers [11]. The registers are also introduced into alternating free model μ-calculus over data trees [12]. Moreover, Figueira proposed an extension of LTL with freeze quantifiers of only one register, where the quantifiers are interpreted over the set of data values occurring before or after the current position of data words [13].
- Recently, Schwentick et al., Demri et al., and Decker et al. considered extensions of LTL with navigation mechanisms for one single data attribute or tuples of data attributes over multi-attributed data words, that is, data words where each position carries multiple data values [14–16].

Another natural idea is to extend temporal logics with variable quantifications over an infinite data domain, which is our focuses in this paper. Grumberg et al. initiated this line of research. They considered the extension of LTL with variable quantifications where the formulae are in *prenex normal form*, that is, all the variable quantifications are in the beginning and followed by LTL formulae. They investigated the decidability of the satisfiability and the model checking problems over Kripke structures extended with data variables [17,18]. Later on, as a follow-up work, they introduced variable CTL* (VCTL*), an extension of CTL* with variable quantifications [19], where the variable quantifications can be nested arbitrarily with Boolean operators and temporal operators. Their main goal is to characterize the simulation pre-order over variable Kripke structures with VCTL* formulae. Nevertheless, as far as we know, they have not investigated the satisfiability and model checking problems of VCTL* yet (except the aforementioned work on VLTL formulae in prenex normal form).

Contribution. Our goal in this paper is to do a relatively complete investigation on the decision problems of the extensions of temporal logics with variable quantifications. Around this purpose, we make the following choices.

- We consider the *extensions of both LTL and CTL* with variable quantifications (denoted by VLTL and VCTL respectively), where the variable quantifiers can be *nested arbitrarily* with Boolean and temporal operators.
- In addition, the variable quantifiers are interpreted over the *full data domain*, not just over the set of data values occurring before or after the current position.
- Moreover, we investigate the decidability and complexity of both the *satisfiability and the model checking* problems over both *finite and infinite* words (trees). More precisely, we consider four decision problems in this paper, the satisfiability and model checking problems (over finite words and trees), the ω-satisfiability and ω-model checking problems (over infinite words and trees).

Specifically, we obtain the following results.

1. Existential variable quantifiers (which may not occur in the beginning) or one single universal quantifier in the beginning already entail the undecidability of the satisfiability and ω-satisfiability problems of both VLTL and VCTL (cf. the results on $\exists^*$-VLTL, $\exists^*$-VCTL, $\forall$-VLTL$_{pnf}$ and $\forall$-VCTL$_{pnf}$ in Table 1).
2. If there are only existential variable quantifiers, and the existential variable quantifiers are not nested, then the satisfiability problem becomes decidable for both VLTL and VCTL (cf. the results on NN-$\exists^*$-VLTL and NN-$\exists^*$-VCTL in Table 1). The proof is obtained by a reduction to the nonemptiness problem of alternating one register automata [13]. On the other hand, in this case, the ω-satisfiability problems of VLTL and VCTL are still undecidable.

Table 1
Summary of the results: U: Undecidable, D: Decidable, P: Proposition, T: Theorem, C: Corollary.

	$\exists^*$-VLTL	$\forall^*$-VLTL	NN-$\exists^*$-VLTL	NN-$\forall^*$-VLTL	$\exists^*$-VLTL$_{pnf}$
SAT	U (T. 4.1)	U (T. 4.2)	D (T. 4.10)	U (T. 4.2)	D (P. 2.5, [17])
ω-SAT	U (T. 4.1)	U (T. 4.2)	U (T. 4.8)	U (T. 4.2)	D (P. 2.5, [17])
MC	U (C. 4.14)	U (C. 4.13)	U (C. 4.14)	D (C. 4.19)	U (C. 4.14)
ω-MC	U (C. 4.14)	U (C. 4.13)	U (C. 4.14)	U (C. 4.17)	U (C. 4.14)

	$\forall^*$-VLTL$_{pnf}$	$\exists$-VLTL$_{pnf}$	$\forall$-VLTL$_{pnf}$	$\exists$-VLTL$^{gdap}_{pnf}$	$\forall$-VLTL$^{gdap}_{pnf}$
SAT	U (T. 4.2)	D (P. 2.5, [17])	U (T. 4.2)	D (P. 2.5, [17])	D (T. 4.11)
ω-SAT	U (T. 4.2)	D (P. 2.5, [17])	U (T. 4.2)	D (P. 2.5, [17])	D (T. 4.11)
MC	D (P. 2.5, [17])	U (C. 4.14)	D (P. 2.5, [17])	D (C. 4.20)	D (P. 2.5, [17])
ω-MC	D (P. 2.5, [17])	U (C. 4.14)	D (P. 2.5, [17])	D (C. 4.20)	D (P. 2.5, [17])

	$\exists\forall$-VLTL$^{noap}_{pnf}$	$\forall\exists$-VLTL$^{noap}_{pnf}$	$\exists\exists$-VLTL$^{noap}_{pnf}$	$\forall\forall$-VLTL$^{noap}_{pnf}$	$\forall\forall\exists$-RVLTL$_{pnf}$
SAT	U (T. 4.7)	U (T. 4.7)	D (P. 2.5, [17])	U (T. 4.7)	U (T. 4.5)
ω-SAT	U (T. 4.7)	U (T. 4.7)	D (P. 2.5, [17])	U (T. 4.7)	U (T. 4.5)
MC	U (C. 4.15)	U (C. 4.15)	U (C. 4.15)	D (P. 2.5, [17])	U (T. 4.16)
ω-MC	U (C. 4.15)	U (C. 4.15)	U (C. 4.15)	D (P. 2.5, [17])	U (T. 4.16)

	$\forall\exists\forall$-RVLTL$_{pnf}$	$\exists^*\forall^*$-RVLTL$_{pnf}$	$\forall^*\exists^*$-RVLTL$_{pnf}$	$\forall\forall$-RVLTL$^{+}_{pnf}$	$\exists\exists$-RVLTL$^{+}_{pnf}$
SAT	U (T. 4.5)	D (T. 4.12)	U (T. 4.5)	U (T. 4.3)	D (P. 2.5, [17])
ω-SAT	U (T. 4.5)	?	U (T. 4.5)	U (T. 4.3)	D (P. 2.5, [17])
MC	U (T. 4.16)	U (T. 4.16)	U (T. 4.16)	D (P. 2.5, [17])	U (T. 4.16)
ω-MC	U (T. 4.16)	U (T. 4.16)	U (T. 4.16)	D (P. 2.5, [17])	U (T. 4.16)

	$\exists^*$-VCTL	$\forall^*$-VCTL	NN-$\exists^*$-VCTL	NN-$\forall^*$-VCTL	$\exists^*$-VCTL$_{pnf}$
SAT	U (C. 5.1)	U (C. 5.2)	D (T. 5.6)	U (C. 5.2)	D (T. 5.15)
ω-SAT	U (C. 5.1)	U (C. 5.2)	U (C. 5.4)	U (C. 5.2)	D (T. 5.15)
MC	U (T. 5.5)	U (T. 5.5)	U (T. 5.5)	D (T. 5.10)	U (T. 5.5)
ω-MC	U (T. 5.5)	U (T. 5.5)	U (T. 5.5)	U (T. 5.5)	U (T. 5.5)

	$\forall^*$-VCTL$_{pnf}$	$\exists$-VCTL$_{pnf}$	$\forall$-VCTL$_{pnf}$	$\exists$-VCTL$^{gdap}_{pnf}$	$\forall$-VCTL$^{gdap}_{pnf}$
SAT	U (C. 5.2)	D (T. 5.15)	U (C. 5.2)	D (T. 5.15)	?
ω-SAT	U (C. 5.2)	D (T. 5.15)	U (C. 5.2)	D (T. 5.15)	?
MC	D (T. 5.16)	?	D (T. 5.16)	?	D (T. 5.16)
ω-MC	D (T. 5.16)	?	D (T. 5.16)	?	D (T. 5.16)

	$\exists\forall$-VCTL$^{noap}_{pnf}$	$\forall\exists$-VCTL$^{noap}_{pnf}$	$\exists\exists$-VCTL$^{noap}_{pnf}$	$\forall\forall$-VCTL$^{noap}_{pnf}$	EVCTL
SAT	U (C. 5.3)	U (C. 5.3)	D (T. 5.15)	U (C. 5.3)	D (T. 5.11)
ω-SAT	U (C. 5.3)	U (C. 5.3)	D (T. 5.15)	U (C. 5.3)	?
MC	U (T. 5.5)	U (T. 5.5)	U (T. 5.5)	D (T. 5.16)	U (T. 5.5)
ω-MC	U (T. 5.5)	U (T. 5.5)	U (T. 5.5)	D (T. 5.16)	U (T. 5.5)

3. If only existential path quantifiers are used in VCTL, then the satisfiability problem is decidable (NEXPTIME), no matter which variable quantifiers are available (cf. the results on EVCTL in Table 1). This result is shown by a small model property. It is open whether the ω-satisfiability problem is decidable in this case.
4. For the fragments of VLTL with one single universal variable quantifier in the beginning, if the occurrences of the non-parameterized atomic propositions are guarded by the positive occurrences of the universally quantified variables, then the satisfiability and ω-satisfiability problems become decidable (cf. the results on $\forall$-VLTL$^{gdap}_{pnf}$ in Table 1). The proof is obtained by a reduction to the nonemptiness of extended data automata [20]. This decidability result is tight in the sense that adding one more existential variable quantifier before or after the universal one implies undecidability (cf. the results on $\forall\exists$-VLTL$^{noap}_{pnf}$ and $\exists\forall$-VLTL$^{noap}_{pnf}$ in Table 1).
5. Moreover, since there are some subtle differences between the logics defined in this paper and those in [17,18], we also investigate the decidability status of the two fragments defined in [17] and [18] and show that some claims in [17,18] are inaccurate (cf. the results on the fragments of RVLTL$_{pnf}$ and RVLTL$^{+}_{pnf}$ in Table 1). In particular, we show that the fragments in [17] behave quite differently from the fragments in [18] with respect to the satisfiability problem.
6. Based on the above results of the satisfiability and ω-satisfiability problems, we deduce the (un)decidability results of the model checking and ω-model checking problems (cf. the results on model checking and ω-model checking problems in Table 1).

For reader's convenience, the results obtained in this paper are summarized into Table 1 (where $\exists, \forall$ mean existential and universal variable quantifier, pnf means prenex normal form, NN means non-nested, and the question mark means that the decidability is open). The reader can refer to Section 2 for the definitions of the fragments of VLTL and VCTL.

This paper is the extended version of the paper published in FSTTCS 2014 [21]. Compared to [21], this paper is novel in the following aspects.

- We add a comparison of the expressiveness of VLTL with the other logical formalisms over (multi-attributed) data words.
- We get more results on the satisfiability and model checking problems, as well as extend the results in [21] to ω-satisfiability and ω-model checking problems.
- We write the detailed proofs for all the results, which were missing or only sketched in [21].

Related work.

First-order temporal logics. At first, we give a more specific description of the work on first-order temporal logics. In [22], Bohn et al. proposed an algorithm for model checking first-order CTL (FO-CTL) on first-order Kripke structures in which transitions are labeled with conditional assignments, capturing the effect of taking a transition on an underlying possibly infinite state space induced from a set of typed variables. Their algorithm is heuristic and may not terminate in some cases. Hodkinson et al. tried to find decidable fragments of first order LTL (FO-LTL) and FO-CTL over natural numbers, and their expressive power and finite axiomatization [7,8]. They showed that the satisfiability problem of *monodic* fragments of FO-LTL and FO-CTL in which all formulae beginning with a temporal operator have at most one free variable is decidable. Moreover, Hodkinson et al. investigated the complexity of the decidable fragments of FO-LTL in [23]. In particular, they proved that the satisfiability problem of one-variable fragment of FO-LTL with the "global" temporal operator is EXPSPACE-complete. Several resolution methods for checking the satisfaction and validity of the formulae in monodic fragments of FO-LTL are proposed in [24–26]. Dixon et al. considered the monodic fragments of FO-LTL with an additional XOR constraint on predicates and showed that with the XOR constraint, a lower complexity upper bound can be obtained for the satisfiability problem [27]. Demri and D'Souza investigated an extension of LTL where the constraints are interpreted over the concrete domains e.g. $(\mathbb{Z}, <, =)$ and $(\mathbb{N}, <, =)$ [28]. Vianu et al. studied the model checking problem of FO-LTL formulae over database driven systems and get some decidability results by putting restrictions on both the systems and the specifications [9,10]. Moreover, Song and Touili considered the variable extensions of LTL and CTL for malware detection, where the variables range over a *finite* domain, although the variable quantifications can be nested arbitrarily with the other operators [29–31].

Indexed temporal logics. Since process identifiers are a concrete type of data values, the indexed temporal logics used to specify and reason about parameterized concurrent systems are also related to the extensions of temporal logic with variable quantifications. Indexed temporal logics are extensions of temporal logics with variable quantifications that range over a set of process identifiers. Browne, Clarke and Grumberg proposed *indexed* CTL$^*\backslash X$ in [32] and proved the *bisimulation* between two Kripke structures with the same set of indexed propositions but different sets of index values with respect to indexed CTL$^*\backslash X$. Emerson and Srinivasan proposed indexed simplified CTL (SCTL) and investigated its satisfiability problem [33]. In [34,35], German and Sistla proposed *indexed* LTL and showed that the validity (resp. model checking) problem of the indexed LTL is decidable (resp. undecidable). Emerson and Kahlon studied the model checking problem of parameterized systems against some specific fragments of *indexed* CTL$^*\backslash X$ [36]. Later, Emerson and Namjoshi also used *indexed* CTL$^*\backslash X$ to specify and reason about parameterized systems in [37]. The main goal of [36,37] is to prove the "cutoff" results. Compared with indexed temporal logics, variables in VLTL and VCTL can range over not only a set of process identifiers but also other data values such as content of messages. On the other hand, each position of the data words/trees for indexed LTL and CTL has the same set of data values, that are process identifies. While, each position of the data words/trees for VLTL and VCTL can have its own set of data values.

Besides the logical formalisms, researchers have also proposed various automata models to reason about data values.

Register automata and its variants. Kaminski and Francez initialized the research of automata models over infinite alphabets. They introduced nondeterministic register automata [38], an extension of finite state automata with a set of registers which can store a symbol from an infinite alphabet. They studied closure properties of nondeterministic register automata and proved that its emptiness problem is decidable. In [39], Grumberg et al. proposed variable (Büchi) automata, a simple extension of finite (Büchi) automata in which the alphabet consists of letters as well as variables that range over the infinite alphabet domain. Variable automata is a sub-type of nondeterministic register automata. Neven et al. studied the expressive power of the variants (one-way v.s. two-way, deterministic v.s. non-deterministic, alternating v.s. non-alternating) of register and pebble automata, and extensions of first-order logic and monadic second-order logic [40]. They proved that universality and containment of one-way nondeterministic register automata and non-emptiness of two-way deterministic register automata are undecidable and that non-emptiness is undecidable even for *weak* one-way deterministic pebble automata. Demri and Lazic introduced alternating one register automata and used it to decide the satisfiability of the fragments of LTL with freeze quantifiers [11]. Kaminski and Zeitlin extended nondeterministic register automata with ϵ-transitions which is able to make a non-deterministic reassignment by "guessing" the content of an appropriate register [41]. Adding ϵ-transitions enriches the expressiveness but still posses all decision procedures and closure properties of nondeterministic register automata. Figueira proposed an extension of alternating one register automata with spread operations and used it to decide the fragments of XPath with data comparison modalities [13].

Data automata and its variants. Data automata were introduced in [42], with the motivation to decide the two-variable first-order logic over data words. Data automata turns out to be a very expressive model for which nonemptiness is decidable and has the same complexity as the reachability of Petri nets. Since it is a famous open problem whether the reachability of Petri nets can be decided with elementary complexity, it is also unknown whether the nonemptiness of data automata can be decided in elementary time. In order to lower the complexity, two weaker versions of data automata were introduced and their nonemptiness problems were shown to be elementary [43,44]. On the other hand, an extension of data automata, called class automata, were introduced, in order to capture the expressiveness of XPath with data comparison modalities [45]. Nevertheless, the nonemptiness of class automata is undecidable. To achieve decidability, two sub-models of class automata: class automata with *priority class condition* and class counting automata, were proposed [46,47]. Tan studied data trees over a linearly ordered infinite data domain and proposed ordered-data tree automata and showed their nonemptiness problem can be solved in 3-NEXPTIME [48].

Pebble automata and its variants. Pebble automata were introduced in [40]. Several variants of this model have been studied. For example, Neven et al. [40] studied alternating and two-way pebble automata. Tan proposed a subclass of pebble automata, *top view weak* pebble automata and showed that the nonemptiness problem is decidable [49]. This model can capture all data languages expressible in LTL with one freeze quantifier. Tan used graph reachability problem to investigate the expressiveness issues of pebble automata, e.g. the strict hierarchy of pebble automata based on the number of pebbles and the comparison of the expressiveness of pebble automata with the other formalisms over infinite alphabets [50].

Outline. The rest of this paper is organized as follows. Preliminaries are given in Section 2. Section 3 compares the expressiveness of VLTL with the other logical formalisms over data words. Section 4 considers the decision problems of VLTL. Section 5 is devoted to the decision problems of VCTL. Conclusion and future work are given in Section 6.

2. Preliminaries

In this section, we first fix some notations, then introduce Variable Kripke Structure (VKS), Variable Linear Temporal Logic (VLTL), Variable Computation Tree Logic (VCTL), alternating register automata and extended data automata. VLTL and VCTL are extensions of LTL and CTL with variables and $(\forall, \exists)$ quantifications. The VCTL and VLTL formulae are interpreted over computation traces and computation trees of variable Kripke structures, respectively. Alternating register automata and extended data automata are used to get the decidability results.

Let $\mathbb{D}$ be an infinite set of data values, AP a finite set of (non-parameterized) atomic propositions, and T with $AP \cap T = \emptyset$ a finite set of parameterized atomic propositions, where each of them carries one parameter (data value). Let $\mathbb{A}$ be a finite set of *attributes*. To get an idea on the data value attributes, let us consider the scenario where a printer is shared by different computers. In this scenario, a "print" event may carry two data parameters which are denoted respectively by the attribute "*cid*" for computer identifiers and the attribute "*tid*" for the identifiers of printing tasks, that is, the set of attributes $\mathbb{A} = \{cid, tid\}$. Let Var be a countable set of data variables which range over $\mathbb{D}$. Let $[k]$ denote the set $\{0, \ldots, k-1\}$, for all $k \in \mathbb{N}$.

2.1. Words and trees

In this paper, we interpret temporal logic formulae over $\mathbb{A}$-attributed data (ω-)words or data (ω-)trees defined in the following.

- A *word* (resp. ω-word) w over AP is a sequence from $(2^{AP})^*$ (resp. $(2^{AP})^\omega$).
- An $\mathbb{A}$-*attributed data word* (resp. data ω-word) w over $AP \cup T$ is a sequence from $(2^{AP} \times (2^T \times \mathbb{D})^{\mathbb{A}})^*$ (resp. $(2^{AP} \times (2^T \times \mathbb{D})^{\mathbb{A}})^\omega$).
- Given $k \geq 1$, a k-ary *tree* (resp. ω-tree) is a finite (resp. infinite) set $Z \subseteq [k]^*$ s.t. for all $zi \in Z$, $z \in Z$ and $zj \in Z$ for all $j \in [i]$ (resp. for all $zi \in Z$, $z \in Z$ and $zj \in Z$ for all $j \in [i]$, moreover, for each $z \in Z$, there is $i \in [k]$ s.t. $zi \in Z$). The node ϵ is called the *root* of the tree. For every $z \in Z$, the nodes $zi \in Z$ for $i \in [k]$ are called the *successors* of z, denoted by $suc(z)$. Let *Leaves*(Z) denote the set of leaves of a tree Z, that is, the set of nodes $z \in Z$ such that $zi \notin Z$ for each $i \in [k]$. A *path* of a tree Z is a set $\pi \subseteq Z$ s.t. $\epsilon \in \pi$ and $\forall z \in \pi$, either z is a leaf, or there is an *unique* $i \in [k]$ s.t. $zi \in \pi$. A *path* of an ω-tree Z is a set $\pi \subseteq Z$ s.t. $\epsilon \in \pi$ and $\forall z \in \pi$, there is an unique $i \in [k]$ s.t. $zi \in \pi$.
- A k-ary *labeled tree* (resp. ω-tree) t over AP is a tuple (Z, L), where Z is a k-ary tree (resp. ω-tree) and $L : Z \to 2^{AP}$ is the labeling function.
- A k-ary $\mathbb{A}$-attributed data tree (resp. ω-tree) t over $AP \cup T$ is a tuple (Z, L), where Z is a k-ary tree (resp. ω-tree) and $L : Z \to 2^{AP} \times (2^T \times \mathbb{D})^{\mathbb{A}}$ is a labeling function.
- Given a labeled or data tree (resp. ω-tree) $t = (Z, L)$, let $z \in Z$ and π be a path of t, then $t|_z$ denotes the labeled or data subtree t rooted at z, and w_π denotes the word or data word (resp. ω-word) on the path π of t.
- For $z \in Z$ in a labeled or data tree (resp. ω-tree) $t = (Z, L)$, define the *tree type* of z in t, denoted by $type_t(z)$, as the set $\{l_0, \ldots, l_{k-1}\}$ s.t. for every $j \in [k]$, if $zj \in Z$, then $l_j = \triangledown_j$, otherwise $l_j = \overline{\triangledown_j}$ ($\triangledown_j$ means that the j-th child of z exists).
- For a data word (or data ω-word) $w = (\alpha_0, (\beta_{a,0}, d_{a,0})_{a \in \mathbb{A}})(\alpha_1, (\beta_{a,1}, d_{a,1})_{a \in \mathbb{A}}) \ldots$, the *projection* of w, denoted by $prj(w)$, is defined as the word $\alpha_0 \alpha_1 \ldots$.

- Let $t = (Z, L)$ and $t' = (Z', L')$ be two k-ary $\mathbb{A}$-attributed data trees over $AP \cup T$. Then an *embedding* of t into t' is an *injective* mapping η from Z to Z' that preserves the root, the successor relation and the labels of nodes, more specifically, $\eta(\varepsilon) = \varepsilon$, for each $z_1, z_2 \in Z$, z_2 is a successor of z_1 in t iff $\eta(z_2)$ is a successor of $\eta(z_1)$ in t', moreover, for each $z \in Z$, $L(z) = L'(\eta(z))$. An embedding η of t into t' is said to be *leaf-preserving* if for each $z \in Z$, z is a leaf in t iff $\eta(z)$ is a leaf in t'. For instance, if t is a k-ary $\mathbb{A}$-attributed data tree with a single node ε, and t' is obtained from t by adding a node 0, then there is a unique embedding of t into t', but the embedding is not leaf-preserving.

In our definition of data (ω-)words and data (ω-)trees, every parameterized atomic proposition is restricted to carry only one parameter (data value). This restriction does not restrict the expressiveness of the formalisms, as it is easy to encode a data (ω-)word or data (ω-)tree where some parameterized atomic propositions have multiple parameters into one satisfying the one-parameter constraint.

2.2. Variable Kripke structure

Definition 2.1 *(Variable Kripke structures).* A *Variable Kripke Structure*[1] (VKS) $\mathcal{K}$ is a tuple $(AP \cup T, X, S, R, S_0, I, L, L')$, where AP and T are defined as above, X and S are finite sets of variables and states respectively, $R \subseteq S \times S$ is the set of edges s.t. for each $s \in S$, there is $s' \in S$ satisfying that $(s, s') \in R$, $S_0 \subseteq S$ is the set of initial states, I is the invariant function that assigns to each state a formula which is a positive Boolean combination of $x_i = x_j$ and $x_i \neq x_j$ for $x_i, x_j \in X$, $L : S \to 2^{AP \cup T \times X}$ is the state labeling function, $L' : R \to 2^{\{reset\} \times X}$ is the edge labeling function.

Intuitively, if $(reset, x) \in L'(s, s')$, then the value of the variable x is reset (to any value) when going from s to s'.

A finite (resp. infinite) *path* of $\mathcal{K}$ is a finite sequence of states $s_0 s_1 \dots s_n$ (resp. an infinite sequence of states $s_0 s_1 \dots$) s.t. $\forall i \in [n]$ (resp. $\forall i \geq 0$), $(s_i, s_{i+1}) \in R$. An X-attributed data word (resp. ω-word) $w_0 w_1 \dots$ is called a finite (resp. infinite) *computation trace* of $\mathcal{K}$ if there are a finite (resp. infinite) path $s_0 s_1 \dots$ in $\mathcal{K}$ with $s_0 \in S_0$ and a finite (resp. infinite) sequence $\lambda_0 \lambda_1 \dots$ s.t.

- $\forall i$, $\lambda_i : X \longrightarrow \mathbb{D}$ is an *assignment function* satisfying that $\lambda_i \models I(s_i)$, and $w_i = (\{p \in AP \mid p \in L(s_i)\}, (\{\tau \mid (\tau, x) \in L(s_i)\} \times \{\lambda_i(x)\})_{x \in X})$, where the satisfaction relation $\lambda_i \models I(s_i)$ is defined in an obvious way, e.g. $\lambda_i \models x = y$ iff $\lambda_i(x) = \lambda_i(y)$,
- for every $i : i > 0$, $\lambda_i(x) = \lambda_{i-1}(x)$ if $(reset, x) \notin L'(s_{i-1}, s_i)$.

Let $\mathcal{L}(\mathcal{K})$ denote the set of finite computation traces of $\mathcal{K}$ and $\mathcal{L}_\omega(\mathcal{K})$ denote the set of infinite computation traces of $\mathcal{K}$.

Let k be the maximum number of successors of states in $\mathcal{K}$. A data tree (resp. ω-tree) $t = (Z, L_1)$ is called a finite (resp. infinite) *computation tree* of $\mathcal{K}$ if there are a k-ary labeled tree (resp. ω-tree) (Z, L_2) over S and a collection of assignment functions $\lambda_z : X \to D$ with $z \in Z$ s.t.

- $\forall z \in Z$, $L_2(z) \in S$ is a singleton,
- $L_2(\varepsilon) \in S_0$,
- $\forall z \in Z \setminus Leaves(Z)$, if $L_2(z) = s$ and s has exactly i successors, say $s_0, \dots, s_{i-1}$, in $\mathcal{K}$, then $suc(z) = \{z0, \dots, z(i-1)\}$, and $\forall j \in [i]$, $L_2(zj) = s_j$,
- for every $z, zi \in Z$, if $L_2(z) = s$ and $L_2(zi) = s'$, then for every $x \in X$ s.t. $(reset, x) \notin L'(s, s')$, $\lambda_{zi}(x) = \lambda_z(x)$,
- for every $z \in Z$, if $L_2(z) = s$, then $L_1(z) = (L(s) \cap AP, (\{\tau \mid (\tau, x) \in L(s)\} \times \lambda_z(x))_{x \in X})$.

Let $\mathcal{T}(\mathcal{K})$ (resp. $\mathcal{T}_\omega(\mathcal{K})$) denote the set of finite (resp. infinite) computation trees of $\mathcal{K}$.

2.3. Variable linear temporal logic

Intuitively, VLTL extends LTL with the modalities $\tau(x)$ and $x@a$ (where $\tau \in T$ and $a \in \mathbb{A}$), besides the existential and universal data variable quantifications. Here $\tau(x)$ means that the parameterized atomic proposition τ holds, with the parameter represented by x, and $x@a$ means that the data value represented by x is the one corresponding to the attribute a in the current position. More specifically, VLTL formulae are defined by the following syntactic rules.

Definition 2.2 *(VLTL).* The syntax of *Variable Linear Temporal Logic* (VLTL) is defined by the following rules,

$$\begin{aligned}\varphi \ := \ & p \mid \neg p \mid \tau(x) \mid \neg\tau(x) \mid x@a \mid \neg x@a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \\ & X\varphi \mid \overline{X}\varphi \mid \varphi \, U \, \varphi \mid \varphi \, R \, \varphi \mid \exists x.\, \varphi \mid \forall x.\, \varphi,\end{aligned}$$

where $p \in AP, \tau \in T, a \in \mathbb{A}, x \in Var$, and $\overline{X}$ is the dual operator of X such that $\overline{X}\varphi \equiv \neg X \neg \varphi$.

[1] Variable Kripke structure defined here is the same as that in [17], except that the global invariants are replaced by local state invariants.

Let $var(\varphi)$ and $free(\varphi)$ denote respectively the set of variables occurring in φ and the set of free variables of φ. VLTL formulae without free variables are called *sentences*. In addition, we use abbreviations $true \equiv p \vee \neg p$, $F\psi \equiv true\ U\ \psi$, and $G\psi \equiv \neg F \neg \psi$.

VLTL formulae defined in [17,18] are in prenex normal form. In addition, VLTL formulae defined in [17,18] allow explicit comparisons between data variables, e.g. $x \neq y$ for two data variables x, y. Moreover, there is a small difference between VLTL defined in [17] and [18]. The atomic formulae τ for $\tau \in T$ are allowed in [17], while they are disallowed in [18]. The formula τ can be seen as an abbreviation of the formula $\exists x.\ \tau(x)$. Therefore, strictly speaking, the formulae in [17] are not in prenex normal form according to our notation. If the explicit data variable comparisons and the formulae τ are ignored, then VLTL defined in [17,18] is the set of VLTL formulae in prenex normal form in our framework. In our definition of VLTL, we make the following technical choices.

- The formulae τ for $\tau \in T$ are disallowed, since we believe that it is a better idea to make all the variable quantifications explicit.
- In addition, the explicit data variable comparisons $x = y$ or $x \neq y$ are disallowed, since we prefer defining a first-order extension of VLTL with *the minimum first-order features* (arguably).

On the other hand, to facilitate the specifications of the properties of $\mathbb{A}$-attributed data words, the formulae $x@a$ are introduced to specify that the data value represented by x is the one corresponding to the attribute a in the current position. The fragments of VLTL defined in [18] and [17], excluding the data variable comparison modalities, are called respectively as RVLTL$_{pnf}$ and RVLTL$^{+}_{pnf}$ in our framework (cf. Section 2.5).

VLTL formulae are interpreted over $\mathbb{A}$-attributed data words and data ω-words. We give the semantics for data words. The semantics for data ω-words are similar.

Let $w = w_0 \dots w_n \in \left(2^{AP} \times (2^T \times \mathbb{D})^{\mathbb{A}}\right)^*$, φ be a VLTL formula, $\lambda : free(\varphi) \to \mathbb{D}$, and for every $i : 0 \le i \le n$, $w_i = (\alpha_i, (\beta_{a,i}, d_{a,i})_{a \in \mathbb{A}})$, and $w^i = w_i \dots w_n$. We define the satisfaction relation $w \models_\lambda \varphi$ as follows:

- $\varphi = p$: $w \models_\lambda \varphi$ iff $p \in \alpha_0$,
- $\varphi = \neg p$: $w \models_\lambda \varphi$ iff $p \notin \alpha_0$,
- $\varphi = \tau(x)$: $w \models_\lambda \varphi$ iff $(\tau, \lambda(x)) \in \cup_{a\in\mathbb{A}}\ (\beta_{a,0} \times \{d_{a,0}\})$,
- $\varphi = \neg\tau(x)$: $w \models_\lambda \varphi$ iff $(\tau, \lambda(x)) \notin \cup_{a\in\mathbb{A}}\ (\beta_{a,0} \times \{d_{a,0}\})$,
- $\varphi = x@a$: $w \models_\lambda \varphi$ iff $d_{a,0} = \lambda(x)$,
- $\varphi = \neg x@a$: $w \models_\lambda \varphi$ iff $d_{a,0} \neq \lambda(x)$,
- $\varphi = \varphi_1 \vee \varphi_2$: $w \models_\lambda \varphi$ iff $w \models_\lambda \varphi_1$ or $w \models_\lambda \varphi_2$,
- $\varphi = \varphi_1 \wedge \varphi_2$: $w \models_\lambda \varphi$ iff $w \models_\lambda \varphi_1$ and $w \models_\lambda \varphi_2$,
- $\varphi = X\varphi_1$: $w \models_\lambda \varphi$ iff $w \neq \varepsilon$ and $w^1 \models_\lambda \varphi_1$,
- $\varphi = \overline{X}\varphi_1$: $w \models_\lambda \varphi$ iff $w \neq \varepsilon$ implies that $w^1 \models_\lambda \varphi_1$,
- $\varphi = \varphi_1\ U\ \varphi_2$: $w \models_\lambda \varphi$ iff there is $i : 0 \le i \le n$ s.t. $w^i \models_\lambda \varphi_2$, and for all $j : 0 \le j < i$, $w^j \models_\lambda \varphi_1$,
- $\varphi = \varphi_1\ R\ \varphi_2$: $w \models_\lambda \varphi$ iff for all $i : 0 \le i \le n$, $w^i \models_\lambda \varphi_2$, or there is $i : 0 \le i \le n$ s.t. $w^i \models_\lambda \varphi_1$, and for all $j : 0 \le j \le i$, $w^j \models_\lambda \varphi_2$,
- $\varphi = \exists x.\ \varphi_1$: $w \models_\lambda \varphi$ iff there is $d \in \mathbb{D}$ s.t. $w \models_{\lambda[d/x]} \varphi_1$, where $(\lambda[d/x])(x) = d$ and $(\lambda[d/x])(y) = \lambda(y)$ for every $y \in free(\varphi_1)$ s.t. $y \neq x$,
- $\varphi = \forall x.\ \varphi_1$: $w \models_\lambda \varphi$ iff for all $d \in \mathbb{D}$, $w \models_{\lambda[d/x]} \varphi_1$.

If φ is a VLTL sentence, we will drop λ from $\models_\lambda$. Let $\mathcal{L}(\varphi)$ (resp. $\mathcal{L}_\omega(\varphi)$) denote the set of $\mathbb{A}$-attributed data words (resp. ω-words) w satisfying φ. Let $\mathcal{K}$ be a VKS with X as the set of variables and φ a VLTL sentence over X-attributed data words. Then $\mathcal{K}$ satisfies φ, denoted by $\mathcal{K} \models \varphi$, if for every finite computation trace w of $\mathcal{K}$, $w \models \varphi$. Similarly, we use $\mathcal{K} \models_\omega \varphi$ to denote the fact that for every infinite computation trace w of $\mathcal{K}$, $w \models \varphi$.

For a VLTL formula φ, let $\overline{\varphi}$ denote the negation of φ, where $\overline{p} = \neg p$, $\overline{\neg p} = p$, $\overline{\tau(x)} = \neg\tau(x)$, $\overline{\neg\tau(x)} = \tau(x)$, $\overline{X\varphi_1} = \overline{X}\overline{\varphi_1}$, $\overline{\varphi_1 U \varphi_2} = \overline{\varphi_1} R \overline{\varphi_2}$, and so on. Let $|\varphi|$ denote the size of φ, that is, the number of symbols in φ. We will use $\varphi_1 \to \varphi_2$ to mean $\overline{\varphi_1} \vee \varphi_2$. A VLTL formula that does not contain subformulae of the form $\psi_1 \to \psi_2$ is called *normalized*.

Let φ be a normalized VLTL formula. For $p_i \in AP$ and an occurrence of p_i (resp. $\neg p_i$) in φ, define the UR-formula of the occurrence of p_i (resp. $\neg p_i$) in φ as the minimal subformula of φ of the form $\psi_1 U \psi_2$ or $\psi_1 R \psi_2$ which contains the occurrence of p_i (resp. $\neg p_i$), if such a subformula exists (otherwise, the UR-formula is undefined). Note that if the UR-formula of an occurrence of p_i or $\neg p_i$ exists, then it is unique. An occurrence of p_i or $\neg p_i$ is said to be *persistent* if the UR-formula of the occurrence exists, and the following condition is satisfied:

- If the UR-formula is of the form $\psi_1 U \psi_2$, then the occurrence of p_i or $\neg p_i$ occurs in ψ_1.
- If the UR-formula is of the form $\psi_1 R \psi_2$, then the occurrence of p_i or $\neg p_i$ occurs in ψ_2.

A non-persistent occurrence of p_i or $\neg p_i$ is called *eventual*. Similarly, we can define the persistent and eventual occurrences for $\tau(x)$ or $\neg\tau(x)$ or $x@a$ or $\neg x@a$ or subformulae of the form $X\psi$. For instance, the occurrence of $\neg p_1$ in the formula $G(\neg p_1 \vee XFp_2)$ is persistent, the occurrence of p_2 is eventual, and the occurrence of XFp_2 is persistent.

2.4. Variable computation tree logic

The syntax of variable computation tree logic is defined similarly to VLTL, by adding the path quantifiers A and E before every temporal operator in the syntax rules of VLTL.

Definition 2.3 *(VCTL).* The syntax of *Variable Computation Tree Logic* (VCTL) formulae is defined by the following rules:

$$\varphi := \begin{array}{c} p \mid \neg p \mid \tau(x) \mid \neg\tau(x) \mid x@a \mid \neg x@a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid AX\varphi \mid EX\varphi \\ \mid A\overline{X}\varphi \mid E\overline{X}\varphi \mid A(\varphi U \varphi) \mid E(\varphi U \varphi) \mid A(\varphi R \varphi) \mid E(\varphi R \varphi) \mid \exists x.\varphi \mid \forall x.\varphi \end{array},$$

where $p \in AP, \tau \in T, a \in \mathbb{A}, x \in Var$.

VCTL formulae are interpreted over $\mathbb{A}$-attributed data trees. Let $t = (Z, L)$ be a k-ary $\mathbb{A}$-attributed data tree with $L(\epsilon) = (\alpha, (\beta_a, d_a)_{a\in\mathbb{A}})$, φ be a VCTL formula, $\lambda : \mathit{free}(\varphi) \to D$, we define the satisfaction relation $(Z, L) \models_\lambda \varphi$ as follows. The semantics of VCTL formulae over $\mathbb{A}$-attributed data ω-trees are similar.

- $\varphi = p$: $t \models_\lambda \varphi$ iff $p \in \alpha$,
- $\varphi = \neg p$: $t \models_\lambda \varphi$ iff $p \notin \alpha$,
- $\varphi = \tau(x)$: $t \models_\lambda \varphi$ iff $(\tau, \lambda(x)) \in \bigcup_{a\in\mathbb{A}} \beta_a \times \{d_a\}$,
- $\varphi = \neg\tau(x)$: $t \models_\lambda \varphi$ iff $(\tau, \lambda(x)) \notin \bigcup_{a\in\mathbb{A}} \beta_a \times \{d_a\}$,
- $\varphi = x@a$: $t \models_\lambda \varphi$ iff $d_a = \lambda(x)$,
- $\varphi = \neg x@a$: $t \models_\lambda \varphi$ iff $d_a \neq \lambda(x)$,
- $\varphi = \varphi_1 \vee \varphi_2$: $t \models_\lambda \varphi$ iff $t \models_\lambda \varphi_1$ or $t \models_\lambda \varphi_2$,
- $\varphi = \varphi_1 \wedge \varphi_2$: $t \models_\lambda \varphi$ iff $t \models_\lambda \varphi_1$ and $t \models_\lambda \varphi_2$,
- $\varphi = AX\varphi_1$: $t \models_\lambda \varphi$ iff $suc(\epsilon) \neq \emptyset$ and $t|_z \models_\lambda \varphi_1$ for all $z \in suc(\epsilon)$,
- $\varphi = EX\varphi_1$: $t \models_\lambda \varphi$ iff $suc(\epsilon) \neq \emptyset$ and $t|_z \models_\lambda \varphi_1$ for some $z \in suc(\epsilon)$,
- $\varphi = A\overline{X}\varphi_1$: $t \models_\lambda \varphi$ iff $suc(\epsilon) \neq \emptyset$ implies that $t|_z \models_\lambda \varphi_1$ for all $z \in suc(\epsilon)$,
- $\varphi = E\overline{X}\varphi_1$: $t \models_\lambda \varphi$ iff $suc(\epsilon) \neq \emptyset$ implies that $t|_z \models_\lambda \varphi_1$ for some $z \in suc(\epsilon)$,
- $\varphi = A[\varphi_1 U \varphi_2]$: $t \models_\lambda \varphi$ iff for every path $\pi \subseteq Z$, there exists a node $z \in \pi$ s.t. $t|_z \models_\lambda \varphi_2$, and for all strict prefixes z' of z, $t|_{z'} \models_\lambda \varphi_1$,
- $\varphi = E[\varphi_1 U \varphi_2]$: $t \models_\lambda \varphi$ iff there is a path $\pi \subseteq Z$ and a node $z \in \pi$ s.t. $t|_z \models_\lambda \varphi_2$, and for all strict prefixes z' of z, $t|_{z'} \models_\lambda \varphi_1$,
- $\varphi = A[\varphi_1 R \varphi_2]$: $t \models_\lambda \varphi$ iff for every path $\pi \subseteq Z$, either $t_z \models_\lambda \varphi_2$ for all $z \in \pi$, or there is $z \in \pi$ s.t. $t|_z \models_\lambda \varphi_1$, and for all prefixes z' of z, $t|_{z'} \models_\lambda \varphi_2$,
- $\varphi = E[\varphi_1 R \varphi_2]$: $t \models_\lambda \varphi$ iff there is a path $\pi \subseteq Z$ s.t. either $t|_z \models_\lambda \varphi_2$ for all $z \in \pi$, or there is $z \in \pi$ s.t. $t|_z \models_\lambda \varphi_1$, and for all prefixes z' of z, $t|_{z'} \models_\lambda \varphi_2$,
- $\varphi = \exists x.\varphi_1$: $t \models_\lambda \varphi$ if there is $d \in \mathbb{D}$ s.t. $t \models_{\lambda[d/x]} \varphi_1$,
- $\varphi = \forall x.\varphi_1$: $t \models_\lambda \varphi$ iff for all $d \in \mathbb{D}$, $t \models_{\lambda[d/x]} \varphi_1$.

Let $\mathcal{L}(\varphi)$ (resp. $\mathcal{L}_\omega(\varphi)$) denote the set of $\mathbb{A}$-attributed data trees (resp. ω-trees) t satisfying φ.

Let $\mathcal{K}$ be a VKS and φ be a VCTL sentence. Then $\mathcal{K}$ satisfies φ, denoted by $\mathcal{K} \models \varphi$, if for every finite computation tree t of $\mathcal{K}$, $t \models \varphi$. Similarly, we use $\mathcal{K} \models_\omega \varphi$ to denote the fact that every infinite computation tree t of $\mathcal{K}$, $t \models_\omega \varphi$.

For a VCTL formula φ, the formula $\overline{\varphi}$ denoting the negation of φ, and $|\varphi|$, the size of φ, can be defined similarly to those of VLTL formulae.

2.5. Syntactic fragments of VLTL and VCTL

In this paper, we consider the following fragments of VLTL.

$\exists^*$-VLTL and $\forall^*$-VLTL. Let $\exists^*$-VLTL (resp. $\forall^*$-VLTL) denote the set of VLTL formulae without using $\forall$ (resp. $\exists$) quantifier. Note that the formulae in $\exists^*$-VLTL and $\forall^*$-VLTL are not required to be in prenex normal form.

NN-VLTL. Let NN-VLTL denote the set of VLTL formulae where no variable quantifiers are nested (in a strict sense), more precisely, for every pair of subformulae $Qx.\psi_1$ and $Q'y.\psi_2$ (where $Q, Q' \in \{\exists, \forall\}$) s.t. $x \neq y$, and $Q'y.\psi_2$ is a subformula of ψ_1, it holds that $x \notin \mathit{free}(\psi_2)$. For instance, the formula $\forall x.(\tau(x) \to \exists y.\tau'(y))$ is in NN-VLTL, while $\forall x.\forall y.\ G[(\tau(x) \wedge \tau'(y)) \to XG(\neg\tau(x) \wedge \neg\tau'(y))]$ is not.

VLTL$_{pnf}$. Let VLTL$_{pnf}$ denote the set of VLTL formulae in prenex normal form $Q_1x_1 \dots Q_kx_k.\ \psi$, where $Q_1, \dots, Q_k \in \{\exists, \forall\}$, and ψ is a quantifier-free VLTL formula. Note that in general VLTL formulae cannot be turned into equivalent prenex normal forms.[2] Suppose $\theta = Q_1 \dots Q_k \in \{\forall, \exists\}^*$, let θ-VLTL$_{pnf}$ denote the set of VLTL$_{pnf}$ formulae of the form $Q_1 x_1 \dots Q_k x_k.\ \psi$. Suppose $\Theta \subseteq \{\forall, \exists\}^*$, let Θ-VLTL$_{pnf} = \cup_{\theta \in \Theta}\ \theta$-VLTL$_{pnf}$.

VLTL$_{pnf}^{noap}$. For $\theta \in \{\exists, \forall\}^*$ (resp. $\Theta \subseteq \{\exists, \forall\}^*$), let θ-VLTL$_{pnf}^{noap}$ (resp. Θ-VLTL$_{pnf}^{noap}$) denote the set of θ-VLTL$_{pnf}$ (resp. Θ-VLTL$_{pnf}$) formulae where $AP = \emptyset$, that is, the set of θ-VLTL$_{pnf}$ (resp. Θ-VLTL$_{pnf}$) formulae containing no occurrence of non-parameterized atomic propositions.

VLTL$_{pnf}^{gdap}$. For $\theta \in \{\exists, \forall\}^*$ (resp. $\Theta \subseteq \{\exists, \forall\}^*$), let θ-VLTL$_{pnf}^{gdap}$ (resp. Θ-VLTL$_{pnf}^{gdap}$) denote the set of θ-VLTL$_{pnf}$ (resp. Θ-VLTL$_{pnf}$) formulae $Q_1 x_1 \dots Q_k x_k.\ \psi$ s.t. $\theta = Q_1 \dots Q_k$ (resp. $Q_1 \dots Q_k \in \Theta$), and all the occurrences of p and $\neg p$ in ψ are *guarded* by the positive occurrences of x. More precisely, ψ is a quantifier-free VLTL formula defined by the following rules,

$$\begin{aligned}\psi \ := \ & p \wedge \tau(x) \mid \neg(p \wedge \tau(x)) \mid \neg p \wedge \tau(x) \mid \neg(\neg p \wedge \tau(x)) \mid p \wedge x@a \mid \\ & \neg(p \wedge x@a) \mid \neg p \wedge x@a \mid \neg(\neg p \wedge x@a) \mid \tau(x) \mid \neg\tau(x) \mid x@a \mid \\ & \neg x@a \mid \psi \vee \psi \mid \psi \wedge \psi \mid X\psi \mid \overline{X}\psi \mid \psi U \psi \mid \psi R \psi,\end{aligned}$$

where $p \in AP$, $\tau \in T$, $a \in \mathbb{A}$, $x \in Var$, and the superscript "*gdap*" means "guarded atomic propositions". For instance, the formula $\forall x.\ G(openFile(x) \to closeFile(x))$ is in $\forall$-VLTL$_{pnf}^{gdap}$, while the formula $\forall x.\ G[openFile(x) \to (p \wedge \neg write(x))\ U\ closeFile(x)]$ is not, since the occurrence of p is not guarded by a positive occurrence of x.

RVLTL$_{pnf}$ and RVLTL$_{pnf}^{+}$. Let RVLTL denote the set of VLTL formulae where the formulae of the form $x@a$ or $\neg x@a$ are not used (The fragment defined in [18], excluding the data variable comparison modalities, corresponds to RVLTL$_{pnf}$). Moreover, in order to facilitate the comparison with the fragments in [17], we use RVLTL$_{pnf}^{+}$ to denote the extension of RVLTL$_{pnf}$ with the formulae τ for $\tau \in T$ which is equivalent to $\exists x.\ \tau(x)$.

Note that by combining the above definitions, more syntactic fragments can be defined. For instance, the fragment NN-$\exists^*$-VLTL is the set of $\exists^*$-VLTL formulae where no variable quantifiers are nested.

The syntactic fragments of VCTL can also be defined similarly to VLTL, with the additional distinction between EVCTL and AVCTL, that is, the fragment of VCTL using only path quantifiers E and A respectively.

2.6. Decision problems of VLTL and VCTL

We consider the following decision problems for VLTL and VCTL.

Satisfiability problem. Given a VLTL (resp. VCTL) sentence φ, decide whether φ is satisfiable, that is, whether there is a data word w (resp. there are $k \geq 1$ and a k-ary $\mathbb{A}$-attributed data tree t) s.t. $w \models \varphi$ (resp. $t \models \varphi$).

ω-Satisfiability problem. Given a VLTL (resp. VCTL) sentence φ, decide whether there is a data ω-word w (resp. there are $k \geq 1$ and a k-ary $\mathbb{A}$-attributed data ω-tree t) s.t. $w \models_\omega \varphi$ (resp. $t \models_\omega \varphi$).

Model checking problem. Given a VKS $\mathcal{K}$ and a VLTL/VCTL sentence φ, decide whether $\mathcal{K} \models \varphi$.

ω-Model checking problem. Given a VKS $\mathcal{K}$ and a VLTL/VCTL sentence φ, decide whether $\mathcal{K} \models_\omega \varphi$.

Remark 2.4. We interpret VLTL and VCTL formulae over both finite and infinite data words (trees). The considerations of temporal logics interpreted over finite words and trees are normally motivated by the verification of safety properties of concurrent systems (cf. [51]) as well as the verification of properties of sequential programs.

The following result is proved essentially in the same way as Theorem 6 in [17]. The main idea is to bound the number of data values satisfying a $\exists^*$-VLTL$_{pnf}$ formula.

Proposition 2.5. *The following problems are PSPACE-complete*[3]*:*

- *the satisfiability and ω-satisfiability problems of $\exists^*$-VLTL$_{pnf}$,*
- *the model checking and ω-model checking problems of $\forall^*$-VLTL$_{pnf}$.*

2.7. Nondeterministic tree automata and nondeterministic Büchi tree automata

A *Nondeterministic Tree Automaton* (NTA) $\mathcal{A}$ is a tuple $(AP, Q, Q_0, \delta, Q_f)$, where AP is a finite set of atomic propositions, Q is a finite set of states, $Q_0, Q_f \subseteq Q$ are the sets of initial and final states, $\delta \subseteq (Q \times 2^{AP}) \cup (Q \times 2^{AP} \times (Q^1 \cup \dots \cup Q^k))$ is the transition relation (a transition $(q, P) \in Q \times 2^{AP}$ means that the current node is a leaf).

[2] For instance, we conjecture that there are no VLTL$_{pnf}$ formulae equivalent to the VLTL formula $G(\exists x.\tau(x))$, but presently we do not know how to prove this.

[3] In [17], only the model checking problem of $\forall^*$-RVLTL$_{pnf}^{+}$ is considered. The results of the (ω-)satisfiability and (ω-)model checking problems of $\exists^*$-VLTL$_{pnf}$ can be shown by following the same idea.

NTAs are used to accept k-ary labeled trees over AP. The semantics of NTA are defined in a standard way. The reader may refer to [52] for the detailed definition. Let $\mathcal{L}(\mathcal{A})$ denote the set of k-ary labeled trees accepted by $\mathcal{A}$.

A *Nondeterministic BÜChi Tree Automaton* (NBTA) $\mathcal{A}$ is a tuple $(AP, Q, Q_0, \delta, Q_f)$, where AP is a finite set of atomic propositions, Q is a finite set of states, $Q_0, Q_f \subseteq Q$ are the sets of initial and accepting states, $\delta \subseteq Q \times 2^{AP} \times (Q^1 \cup \cdots \cup Q^k)$ is the transition relation. NBTAs are used to accept k-ary labeled ω-trees over AP. Let t be a labeled ω-tree t over AP. A run of an NBTA $\mathcal{A}$ over t can be defined in a natural way. A run is *accepting* iff over each infinite path of the run, there is a state from Q_f occurring infinitely often. Let $\mathcal{L}(\mathcal{A})$ denote the set of k-ary labeled ω-trees accepted by $\mathcal{A}$.

Proposition 2.6. *([52,53]) The nonemptiness of NTAs (resp. NBTAs) is in PTIME.*

2.8. Alternating register automata

We next define alternating register automata over k-ary $\mathbb{A}$-attributed data trees where $\mathbb{A}$ is a singleton by adapting the definition of alternating register automata over data (ω-)words and unranked data trees [11,13].

In this subsection, we always assume that $\mathbb{A}$ is a singleton.

Definition 2.7 *(Alternating register automata).* An *Alternating Register Automaton* (ATRA) over k-ary $\mathbb{A}$-attributed data trees where $\mathbb{A}$ is a singleton is a tuple $\mathcal{A} = (AP \cup T, Q, q_0, \delta)$, where AP (resp. T) is a finite set of atomic propositions (resp. parameterized atomic propositions), Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \to \Phi$ is the transition function, where Φ is defined by the following grammar,[4]

$$true \mid false \mid p \mid \neg p \mid \tau \mid \neg\tau \mid \triangledown_i? \mid \overline{\triangledown_i}? \mid eq \mid \overline{eq} \mid q \vee q' \mid q \wedge q' \mid store(q) \mid guess(q) \mid \triangledown_i q,$$

where $p \in AP$, $\tau \in T$, $q, q' \in Q$, and $i \in [k]$.

Intuitively, $p, \neg p, \tau, \neg\tau$ are used to detect the occurrences of (parameterized) atomic propositions. $\triangledown_i?, \overline{\triangledown_i}?$ are used to describe the types of nodes in trees, $eq, \overline{eq}$ are used to check whether the data value in the register is equal to the current one, $q \vee q'$ makes a nondeterministic choice, $q \wedge q'$ creates two threads with the state q and q' respectively, $store(q)$ stores the current data value (note that $\mathbb{A}$ is a singleton and there is exactly one data value over each position) to the register and transfers to the state q, $guess(q)$ guesses a data value for the register and transfers to the state q, $\triangledown_i q$ moves to the i-th child of the current node and transfers to the state q.

An ATRA $\mathcal{A} = (AP \cup T, Q, q_0, \delta)$ is called *alternating register automaton* over $\mathbb{A}$-attributed data words (AWRA) where $\mathbb{A}$ is a singleton if $k = 1$.

The semantics of ATRAs over k-ary $\mathbb{A}$-attributed data trees where $\mathbb{A}$ is a singleton are defined in a completely analogous way as those of ATRAs over unranked trees in [13]. To make the paper more self-contained, we describe the semantics of ATRAs in the following.

Let $\mathcal{A}$ be an ATRA and $t = (Z, L)$ be a k-ary $\mathbb{A}$-attributed data tree. A *node configuration* c of $\mathcal{A}$ is a tuple $(z, type_t(z), L(z), \Lambda)$, where $z \in Z$, $\Lambda \subseteq Q \times \mathbb{D}$ is a finite set of active threads at node z in which each thread (q, d) denotes that the thread is at state q and has the data value d, satisfying the following condition: For every $(q, d) \in \Lambda$ s.t. $\delta(q) = \triangledown_i q'$ for some $q' \in Q$, we have $\triangledown_i \in type_t(z)$. A *tree configuration* C of $\mathcal{A}$ is a finite set of node configurations. Let $N_{\mathcal{A}}$ denote the set of node configurations of $\mathcal{A}$, and $T_{\mathcal{A}} \subseteq 2^{N_{\mathcal{A}}}$ be the set of tree configurations. A tree configuration C is called *initial* if $C = \{(\varepsilon, type_t(\varepsilon), L(\varepsilon), \{(q_0, d)\})\}$ s.t. $L(\varepsilon) = (A, (B, d))$ for some $A \subseteq AP$ and $B \subseteq T$. To define a run of $\mathcal{A}$, we introduce two types of transition relations, the *non-moving* relation $\longrightarrow_\epsilon \subseteq N_{\mathcal{A}} \times N_{\mathcal{A}}$ and the *moving* relations $\longrightarrow_{\triangledown_i} \subseteq N_{\mathcal{A}} \times N_{\mathcal{A}}$ for $i \in [k]$. For a given node configuration $c = (z, type_t(z), L(z), \{(q, d)\} \cup \Lambda)$, the non-moving relation updates a thread (q, d) of c according to the transition function $\delta(q)$, and does not move to any child of z in t. Formally, $\longrightarrow_\epsilon \subseteq N_{\mathcal{A}} \times N_{\mathcal{A}}$ is defined as follows,

- $(z, type_t(z), L(z), \{(q, d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \Lambda)$, if $\delta(q) = true$;
- $(z, type_t(z), L(z), \{(q, d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \Lambda)$, if $\delta(q) = p$, and there exist $A \subseteq AP$, $B \subseteq T$, and $d' \in \mathbb{D}$ such that $L(z) = (A, (B, d'))$ and $p \in A$;
- $(z, type_t(z), L(z), \{(q, d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \Lambda)$, if $\delta(q) = \neg p$, and there exist $A \subseteq AP$, $B \subseteq T$, and $d' \in \mathbb{D}$ such that $L(z) = (A, (B, d'))$ and $p \notin A$;
- $(z, type_t(z), L(z), \{(q, d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \Lambda)$, if $\delta(q) = \tau$ and there exist $A \subseteq AP$, $B \subseteq T$, and $d' \in \mathbb{D}$ such that $L(z) = (A, (B, d'))$ and $\tau \in B$;
- $(z, type_t(z), L(z), \{(q, d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \Lambda)$, if $\delta(q) = \neg\tau$ and there exist $A \subseteq AP$, $B \subseteq T$, and $d' \in \mathbb{D}$ such that $L(z) = (A, (B, d'))$ and $\tau \notin B$;
- $(z, type_t(z), L(z), \{(q, d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \Lambda)$, if $\delta(q) = \triangledown_i?$ and $zi \in Z$;
- $(z, type_t(z), L(z), \{(q, d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \Lambda)$, if $\delta(q) = \overline{\triangledown_i}?$ and $zi \notin Z$;

[4] The *spread* mechanism in [13] is dropped, since it is not used in this paper.

- $(z, type_t(z), L(z), \{(q,d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \Lambda)$, if $\delta(q) = eq$ and there exist $A \subseteq AP$ and $B \subseteq T$ such that $L(z) = (A, (B, d))$;
- $(z, type_t(z), L(z), \{(q,d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \Lambda)$, if $\delta(q) = \overline{eq}$ and there exist $A \subseteq AP$, $B \subseteq T$ and $d' \in \mathbb{D}$ such that $L(z) = (A, (B, d'))$ and $d \neq d'$;
- for $j = 1, 2$, $(z, type_t(z), L(z), \{(q,d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \{(q_j, d)\} \cup \Lambda)$ if $\delta(q) = q_1 \vee q_2$;
- $(z, type_t(z), L(z), \{(q,d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \{(q_1, d), (q_2, d)\} \cup \Lambda)$, if $\delta(q) = q_1 \wedge q_2$;
- $(z, type_t(z), L(z), \{(q,d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \{(q', d')\} \cup \Lambda)$, if $\delta(q) = store(q')$ and there exist $A \subseteq AP$ and $B \subseteq T$ such that $L(z) = (A, (B, d'))$;
- $(z, type_t(z), L(z), \{(q,d)\} \cup \Lambda) \longrightarrow_\epsilon (z, type_t(z), L(z), \{(q', d')\} \cup \Lambda)$ for all $d' \in \mathbb{D}$, if $\delta(q) = guess(q')$.

A node configuration $(z, type_t(z), (A, B), \Lambda)$ is *moving* if for every $(q, d) \in \Lambda$, we have $\delta(q) = \triangledown_i q'$ for some $i \in [k]$. The moving relations $\longrightarrow_{\triangledown_i}$ advance some threads of a moving node configuration to the i-th child. Suppose $i \in [k]$ and $(z, type_t(z), L(z), \Lambda)$ is a moving node configuration s.t. $\triangledown_i \in type_t(z)$. Then

$$(z, type_t(z), L(z), \Lambda) \longrightarrow_{\triangledown_i} (zi, type_t(zi), L(zi), \Lambda'),$$

where $\Lambda' = \{(q', d) \mid (q, d) \in \Lambda, \delta(q) = \triangledown_i q'\}$. Note that Λ' may be $\emptyset$ if there are no $(q, d) \in \Lambda$ such that $\delta(q) = \triangledown_i q'$.

The transition relation $\longrightarrow$ of tree configurations is defined as follows. Let C_1, C_2 be two tree configurations. Then $C_1 \longrightarrow C_2$ if one of the following conditions holds:

- $C_1 = \{c\} \cup C'$ and $C_2 = \{c'\} \cup C'$ s.t. $c \to_\varepsilon c'$.
- $C_1 = \{c\} \cup C'$, $c = (z, type_t(z), L(z), \Lambda)$, $type_t(z) = \{\triangledown_0, \ldots, \triangledown_i, \overline{\triangledown_{i+1}}, \ldots, \overline{\triangledown_{k-1}}\}$ for some $i \in [k]$, there is no $(q, d) \in \Lambda$ s.t. $\delta(q) = \triangledown_j q'$ for $j : i < j < k$ and $q' \in Q$, and $C_2 = \{c'_0, \ldots, c'_i\} \cup C'$ s.t. $c \to_{\triangledown_j} c'_j$ for every $j : 0 \le j \le i$.

A *run* of $\mathcal{A}$ over a data tree $t = (Z, L)$ is a sequence of tree configurations $C_0 \ldots C_n$ s.t. C_0 is initial and for all $i : 1 \le i \le n$, $C_{i-1} \longrightarrow C_i$. A run $C_0 \ldots C_n$ is *accepting* if $C_n \subseteq \{(z, type_t(z), L(z), \emptyset) \mid z \in Z\}$. A data tree $t = (Z, L)$ is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ over t. Let $\mathcal{L}(\mathcal{A})$ denote the set of all k-ary $\mathbb{A}$-attributed data trees accepted by $\mathcal{A}$.

The closure properties and the decidability of the nonemptiness of ATRAs over finite words can be proved in the same way as alternating register automata over unranked trees, by utilizing well-structured transition systems (cf. [13]).

Theorem 2.8 *([11,13]). ATRAs are closed under intersection and union. The nonemptiness problem of ATRAs is decidable and non-primitive recursive.*

2.9. Extended data automata

We also assume that $\mathbb{A} = \{a\}$ is a singleton in this subsection and introduce extended data automata [20], another automata model over ($\mathbb{A}$-attributed) data (ω-)words. Extended data automata are an extension of the seminal model of data automata over data (ω-)words [54].

Definition 2.9 *(Extended data automata).* An *Extended Data Automaton* (EDA) $\mathcal{D}$ over $\mathbb{A}$-attributed data words is a tuple $(AP \cup T, \mathcal{A}, \mathcal{B})$ s.t. AP and T are defined as above, $\mathcal{A}$ is a nondeterministic letter-to-letter transducer over finite words from the alphabet $2^{AP} \times 2^T$ to some output alphabet Σ, and $\mathcal{B}$ is a finite automaton over $\Sigma \cup \{0\}$ (where $0 \notin \Sigma$). On the other hand, an extended data automaton $\mathcal{D}$ over $\mathbb{A}$-attributed data ω-words (abbreviated as ω-EDA) is a tuple $(AP \cup T, \mathcal{A}, \mathcal{B})$ where $\mathcal{A}$ is a nondeterministic letter-to-letter transducer over ω-words from the alphabet $2^{AP} \times 2^T$ to some output alphabet Σ, and $\mathcal{B}$ is a Büchi automaton over $\Sigma \cup \{0\}$.

Let $\mathcal{D} = (AP \cup T, \mathcal{A}, \mathcal{B})$ be an EDA and $w = (\alpha_0, (\beta_{a,0}, d_{a,0})) \ldots (\alpha_n, (\beta_{a,n}, d_{a,n}))$ be an $\mathbb{A}$-attributed data word. Then w is accepted by $\mathcal{D}$ if over $(\alpha_0, \beta_{a,0}) \ldots (\alpha_n, \beta_{a,n})$, the transducer $\mathcal{A}$ outputs a word $w' = \sigma_0 \ldots \sigma_n$ over the alphabet Σ, s.t. for every data value $d \in \mathbb{D}$, $cstr_d(w'')$ is accepted by $\mathcal{B}$, where $w'' = (\sigma_0, d_{a,0}) \ldots (\sigma_n, d_{a,n})$ and $cstr_d(w'')$ is defined as $\sigma'_0 \ldots \sigma'_n$, satisfying that for every $i : 0 \le i \le n$, $\sigma'_i = \sigma_i$ if $d_{a,i} = d$, and $\sigma'_i = 0$ otherwise. Note that for every data value d not occurring in w'', $cstr_d(w'') = 0^{n+1}$.

Let $\mathcal{D} = (AP \cup T, \mathcal{A}, \mathcal{B})$ be an ω-EDA and $w = (\alpha_0, (\beta_{a,0}, d_{a,0}))(\alpha_1, (\beta_{a,1}, d_{a,1})) \ldots$ be an $\mathbb{A}$-attributed data ω-word. Then w is accepted by $\mathcal{D}$ if over $(\alpha_0, \beta_{a,0})(\alpha_1, \beta_{a,1}) \ldots$, $\mathcal{A}$ outputs an ω-word $w' = \sigma_0\sigma_1 \ldots$ over the alphabet Σ, s.t. for every data value $d \in \mathbb{D}$, $cstr_d(w'')$ is accepted by $\mathcal{B}$, where $w'' = (\sigma_0, d_{a,0})(\sigma_1, d_{a,1}) \ldots$ and $cstr_d(w'')$ is defined as $\sigma'_0\sigma'_1 \ldots$, satisfying that for every $i : i \ge 0$, $\sigma'_i = \sigma_i$ if $d_{a,i} = d$, and $\sigma'_i = 0$ otherwise. Note that for every data value d not occurring in w, $cstr_d(w'') = 0^\omega$.

Theorem 2.10. *([54,20]) EDAs and ω-EDAs are closed under intersection and union. The nonemptiness problems of EDAs and ω-EDAs are decidable.*

3. Expressiveness

In the following, we compare the expressiveness of VLTL with first-order logic (FO), freeze LTL [55] and BDLTL [14,16] over $\mathbb{A}$-attributed data words.

3.1. Comparison with first-order logic

The syntax of first-order logic over $\mathbb{A}$-attributed data words is defined by the following rules,

$$\begin{aligned}\varphi \ := \ & p(x) \mid \neg p(x) \mid \tau(a@x) \mid \neg\tau(a@x) \mid a@x = b@y \mid \neg\ a@x = b@y \\ & \mid x = y \mid x \neq y \mid x < y \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\ \varphi \mid \forall x.\ \varphi,\end{aligned}$$

where $p \in AP, \tau \in T, a, b \in \mathbb{A}$.

The atomic formula $\tau(a@x)$ states that (τ, d) occurs in the position x, where d is the data value corresponding to the attribute a. The atomic formula $a@x = b@y$ states that the data value d and d' are the same, where d is the data value corresponding to the attribute a of the position x, and d' is the data value corresponding to the attribute b of the position y.

Proposition 3.1. *VLTL $\leq$ FO.*

Proof. Without loss of generality, we assume that for every VLTL formula φ, no variables in φ are quantified twice. Let $x_1, \ldots, x_n$ be the set of variables occurring in φ, and $x'_1, \ldots, x'_n$ be a tuple of variables distinct from $x_1, \ldots, x_n$.

In the following, we provide a translation of the VLTL formula φ into a FO formula ψ. The intuition of the translation is to replace the quantifiers over the data domain by the quantifiers over the positions in the data words. For each variable quantifier $\exists x.\varphi_1$, we distinguish between the situation that the data value assigned to x will occur hereafter or not. If the former situation happens, the reference to the data value assigned to x can be replaced by a reference to some future position x' in data words.

By induction on the structure of VLTL formulae, for a given VLTL formula φ with the set of free variables $\{x_{i_1}, \ldots, x_{i_k}\}$ (where $1 \leq i_1, \ldots, i_k \leq n$), we translate φ into a FO formula $tr_\eta(\varphi)(z)$ as follows, where η is a function assigning $a_j@x'_{i_j}$ for some $a_j \in \mathbb{A}$ to x_{i_j} for each $j : 1 \leq j \leq k$.

- $tr_\eta(p)(z) = p(z)$, $tr_\eta(\neg p)(z) = \neg p(z)$,
- $tr_\eta(\tau(x))(z) = \vee_{a\in\mathbb{A}}(\eta(x) = a@z \wedge \tau(a@z))$,
- $tr_\eta(\neg\tau(x))(z) = \wedge_{a\in\mathbb{A}}(\neg\eta(x) = a@z \vee \neg\tau(a@z))$,
- $tr_\eta(\varphi_1 \vee \varphi_2)(z) = tr_\eta(\varphi_1)(z) \vee tr_\eta(\varphi_2)(z)$, $tr_\eta(\varphi_1 \wedge \varphi_2)(z) = tr_\eta(\varphi_1)(z) \wedge tr_\eta(\varphi_2)(z)$,
- $tr_\eta(X\varphi_1)(z) = \exists y.\ y = z + 1 \wedge tr_\eta(\varphi_1)(y)$,
- $tr_\eta(\overline{X}\varphi_1)(z) = \forall y.\ y = z + 1 \rightarrow tr_\eta(\varphi_1)(y)$,
- $tr_\eta(\varphi_1\ U\ \varphi_2)(z) = \exists y.\ z \leq y \wedge tr_\eta(\varphi_2)(y) \wedge \forall z'.\ (z \leq z' \wedge z' < y) \rightarrow tr_\eta(\varphi_1)(z')$,
- $tr_\eta(\exists x_i.\varphi_1)(z) = tr_\eta(\varphi'_1)(z) \vee \vee_{a\in\mathbb{A}}\exists x'_i.\ (z \leq x'_i \wedge tr_{\eta[a@x'_i/x_i]}(\varphi_1)(z))$, where φ'_1 is obtained from φ_1 by replacing each occurrence of $\tau(x_i)$ and $x_i@a'$ for $\tau \in T$ and $a' \in \mathbb{A}$ with *false*,
- $tr_\eta(\forall x_i.\varphi_1)(z) = tr_\eta(\varphi'_1)(z) \wedge \wedge_{a\in\mathbb{A}}\forall x'_i.\ (z \leq x'_i \rightarrow tr_{\eta[a@x'_i/x_i]}(\varphi_1)(z))$, where φ'_1 is obtained from φ_1 by replacing each occurrence of $\tau(x_i)$ and $x_i@a'$ for $\tau \in T$ and $a' \in \mathbb{A}$ with *false*.

Claim. Let w be an $\mathbb{A}$-attributed data word, φ be a VLTL formula, $\lambda : free(\varphi) \rightarrow \mathbb{D}$, η be a function with the domain $free(\varphi)$ such that for each $x \in free(\varphi)$, $\eta(x) = a@x'$ for some $a \in \mathbb{A}$, $\lambda' : \{x' \mid x \in free(\varphi)\} \rightarrow \mathbb{N}$. In addition, for each $x \in free(\varphi)$ s.t. $\eta(x) = a@x'$, $\lambda(x)$ is equal to the data value corresponding to the attribute a of the position $\lambda'(x')$ in w. Then $w \models_\lambda \varphi$ iff $w \models_{\lambda'[0/z]} tr_\eta(\varphi)$.

Then a VLTL sentence φ is equal to the FO formula $\exists z.(\forall z'.z \leq z') \wedge tr(\varphi)(z)$, where η is omitted since its domain is empty. $\square$

3.2. Comparison with freeze LTL

A fragment of LTL with freeze quantifiers (Freeze-LTL) over $\mathbb{A}$-attributed data words was considered in [55]. Freeze-LTL is defined by ignoring the parameterized atomic propositions, that is, $T = \emptyset$, and adding a finite set of registers $\mathcal{R}$. Freeze-LTL formulae are defined by the following rules,

$$\varphi ::= p \mid \neg p \mid \downarrow_r^a \varphi \mid \uparrow_r^{\sim a} \mid \uparrow_r^{\nsim a} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid \overline{X}\varphi \mid \varphi U \varphi \mid \varphi R \varphi,$$

where $a \in \mathbb{A}$ and $r \in \mathcal{R}$.

The atomic formula $\downarrow_r^a \varphi$ is used to store the data value corresponding to the attribute a in the current position into the register r, and $\uparrow_r^{\sim a}$ (resp. $\uparrow_r^{\nsim a}$) states that the data value corresponding to the attribute a is equal (resp. not equal) to that stored in the register r.

Proposition 3.2. *Freeze-LTL $\leq$ VLTL.*

Proof. For each freeze-LTL formula φ, an equivalent VLTL formula $tr(\varphi)$ can be inductively constructed as follows, where a distinct data variable x_r is associated for each $r \in \mathcal{R}$,

- $tr(p) = p$, $tr(\neg p) = \neg p$,
- $tr(\downarrow_r^a \varphi_1) = \exists x_r.\ x_r@a \wedge tr(\varphi_1)$,
- $tr(\uparrow_r^{\sim a}) = x_r@a$, $tr(\uparrow_r^{\nsim a}) = \neg x_r@a$,
- $tr(\varphi_1 \wedge \varphi_2) = tr(\varphi_1) \wedge tr(\varphi_2)$, $tr(\varphi_1 \vee \varphi_2) = tr(\varphi_1) \vee tr(\varphi_2)$,
- $tr(X\varphi_1) = Xtr(\varphi_1)$, $tr(\overline{X}\varphi_1) = \overline{X}tr(\varphi_1)$,
- $tr(\varphi_1 U \varphi_2) = tr(\varphi_1)\ U\ tr(\varphi_2)$,
- $tr(\varphi_1 R \varphi_2) = tr(\varphi_1)\ R\ tr(\varphi_2)$. □

3.3. Comparison with BDLTL

BDLTL formulae from [14,16] are defined by ignoring the parameterized atomic propositions, that is, $T = \emptyset$.
The syntax of BDLTL is defined by the following rules,

$$\begin{aligned} \varphi &::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid \overline{X}\varphi \mid \varphi U \varphi \mid \varphi R \varphi \mid C_a^i \psi, \\ \psi &::= @a \mid \neg @a \mid \psi \wedge \psi \mid \psi \vee \psi \mid X^= \psi \mid \overline{X}^= \psi \mid \psi U^= \psi \mid \psi R^= \psi \mid \varphi, \end{aligned}$$

where $p \in AP, i \in \mathbb{N}, a \in \mathbb{A}$. The formulae φ are called the state formulae, while the formulae ψ are called the class formulae.
Intuitively, $C_a^i \psi$ means that the data value d corresponding to the attribute a is stored into the (unique) register and the class formula ψ holds in the i-th position after the current position, under the context that the register stores the data value d. The class formula $@a$ (resp. $\neg @a$) states that the data value corresponding to the attribute a is equal (resp. not equal) to that stored in the register. The class formula $X^= \psi$ requires that ψ holds in the position corresponding to the next occurrence of the data value d. Similarly for $\overline{X}^= \psi$. The formula $\psi_1 U^= \psi_2$ describes the fact that ψ_2 holds in some future position i where the data value d occurs, and in each position $j : 0 \leq j < i$ where the data value d occurs, ψ_1 holds. Similarly for $\psi_1 R^= \psi_2$.

Remark 3.3. Since VLTL is defined without past temporal operators, we consider BDLTL with only future temporal operators here.

Proposition 3.4. *$BDLTL < VLTL$.*

Proof. Each BDLTL state formula φ can be translated into a VLTL formula. The translation is obvious, except for the rule $C_a^i \psi$ and the class formulae. The formula $C_a^i \psi$ can be translated into the formula $\exists x.\ x@a \wedge X^i\ tr_x(\psi)$, where $tr_x(\psi)$ is defined inductively as follows,

- $tr_x(@a) = x@a$, $tr_x(\neg @a) = \neg x@a$,
- $tr_x(\psi_1 \wedge \psi_2) = tr_x(\psi_1) \wedge tr_x(\psi_2)$, $tr_x(\psi_1 \vee \psi_2) = tr_x(\psi_1) \vee tr_x(\psi_2)$,
- $tr_x(X^= \psi_1) = X\,((\wedge_{a \in \mathbb{A}} (\neg x@a))\ U\ (\vee_{a \in \mathbb{A}} x@a \wedge tr_x(\psi_1)))$,
- $tr_x(\overline{X}^= \psi_1) = X\mathit{false} \vee tr_x(X^= \psi_1)$,
- $tr_x(\psi_1 U^= \psi_2) = ((\vee_{a \in \mathbb{A}} x@a) \rightarrow tr_x(\psi_1))\ \ U\ \ (\vee_{a \in \mathbb{A}} x@a \wedge tr_x(\psi_2))$,
- $tr_x(\psi_1 R^= \psi_2) = ((\vee_{a \in \mathbb{A}} x@a) \wedge tr_x(\psi_1))\ \ R\ \ ((\vee_{a \in \mathbb{A}} x@a) \rightarrow tr_x(\psi_2))$.

The argument for the strictness of the inclusion is as follows: Consider the $\exists^*$-*VLTL* formula φ in Theorem 4.1 that expresses the solution of the PCP problem, φ cannot be expressed in any BDLTL formula due to the fact that the satisfiability for BDLTL is decidable [14]. □

4. Decision problems of VLTL

4.1. Satisfiability problem

4.1.1. Undecidability

In this subsection, we will present the undecidability results of the satisfiability and ω-satisfiability problems for various fragments of VLTL. We will only do the proofs for the satisfiability problem, and it is easy to see that the undecidability proofs carry over to the ω-satisfiability problem.

Theorem 4.1. *The satisfiability and ω-satisfiability problems of $\exists^*$-VLTL are undecidable.*

Proof. The proof is by a reduction from the PCP problem.

Let $(u_i, v_i)_{1 \le i \le n}$ be an instance of the PCP problem over an alphabet Σ. A solution of the PCP problem is a sequence of indexes $i_1 \dots i_m$ s.t. $u_{i_1} \dots u_{i_m} = v_{i_1} \dots v_{i_m}$.

During the proof, we will use the following alphabet, $\Sigma' = \Sigma \cup \{\overline{a} \mid a \in \Sigma\} \cup \{1, \dots, n\} \cup \{\overline{1}, \dots, \overline{n}\} \cup \{\#\}$.

The alphabet Σ' can be encoded by $\lceil \log(|\Sigma'|) \rceil$ bits. Let AP be a set of atomic propositions of size $\lceil \log(|\Sigma'|) \rceil$. For each $\sigma \in \Sigma'$, let $atom(\sigma)$ denote the element of 2^{AP} corresponding to the encoding of σ, and $type(\sigma)$ denote the conjunction of atomic propositions or negated atomic propositions from AP corresponding to the binary encoding of σ. For instance, if σ is encoded by 10 and $AP = \{p_1, p_2\}$, then $atom(\sigma) = \{p_1\}$ and $type(\sigma) = p_1 \wedge \neg p_2$. The definition of $atom(\sigma)$ can be naturally extended to $atom(u)$ for words $u \in (\Sigma')^+$. In addition, let $T = \{\tau\}$ and $\mathbb{A} = \{a\}$.

We intend to encode a solution of the PCP problem, say $i_1 \dots i_m$, as an $\mathbb{A}$-attributed data word of the form $w_{i_1} w_{i_2} \dots w_{i_m} (atom(\#), (\{\tau\}, d))\ \overline{w_{i_1}} \dots \overline{w_{i_m}}$ s.t.

- $prj(w_{i_j}) = atom(i_j)\ atom(u_{i_j})$ and $prj(\overline{w_{i_j}}) = atom(\overline{i_j})\ atom(\overline{v_{i_j}})$ for every $j : 1 \le j \le m$,
- the two sequences of data values in the positions corresponding to respectively $atom(i_1) \dots atom(i_m)$ and $atom(\overline{i_1}) \dots atom(\overline{i_m})$ are the same,
- the two sequences of data values in the positions corresponding to respectively $atom(u_{i_1}) \dots atom(u_{i_m})$ and $atom(\overline{v_{i_1}}) \dots atom(\overline{v_{i_m}})$ are the same.

The $\mathbb{A}$-attributed data words satisfying the above conditions can be expressed by the $\exists^*$-VLTL formula φ which is the conjunction of the following $\exists^*$-VLTL formulae.

- There is only one occurrence of $atom(\#)$ in the data word,

$$\varphi_1 = F\ type(\#) \wedge G(type(\#) \rightarrow XG\neg type(\#)).$$

- For every $j : 1 \le j \le n$ (resp. $\bar{j}$), every occurrence of $atom(j)$ (resp. $atom(\bar{j})$) is followed by $atom(u_j)$ (resp. $atom(\overline{v_j})$),

$$\varphi_2 = \bigwedge_{1 \le j \le n} G(type(j) \rightarrow X\psi_{u_j}) \wedge G(type(\bar{j}) \rightarrow X\psi_{\overline{v_j}}),$$

where ψ_{u_j} is the VLTL formula expressing that $atom(u_j)$ will occur in the next $|u_j|$ positions and will be followed by another letter from $\{atom(1), \dots, atom(n)\}$ or $atom(\#)$. For instance, if $u_j = \sigma_1\sigma_2\sigma_1$, then

$$\psi_{u_j} = type(\sigma_1) \wedge Xtype(\sigma_2) \wedge XX\left(type(\sigma_1) \wedge \left(X \bigvee_{1 \le j' \le n} type(j') \vee Xtype(\#)\right)\right).$$

Similarly for $\psi_{\overline{v_j}}$, where $Xtype(\#)$ is replaced by $\overline{X}false$.

- No data values in two positions labeled by letters from $\{atom(1), \dots, atom(n)\}$ (resp. $\{atom(\overline{1}), \dots, atom(\overline{n})\}$) are the same,

$$\varphi_3 = \bigwedge_{1 \le j \le n} \left(\begin{array}{l} G\left(\begin{array}{l} type(j) \rightarrow \\ \exists x. \left(type(j) \wedge \tau(x) \wedge \overline{X}G\left(\bigwedge_{1 \le j' \le n} (\neg type(j') \vee \neg\tau(x)) \right)\right) \end{array} \right) \\ \wedge G\left(\begin{array}{l} type(\bar{j}) \rightarrow \\ \exists x. \left(type(\bar{j}) \wedge \tau(x) \wedge \overline{X}G\left(\bigwedge_{1 \le j' \le n} (\neg type(\bar{j'}) \vee \neg\tau(x)) \right)\right) \end{array} \right) \end{array} \right).$$

- No data values in two positions labeled by letters from $\{atom(\sigma) \mid \sigma \in \Sigma\}$ (resp. letters from $\{atom(\overline{\sigma}) \mid \sigma \in \Sigma\}$) are the same, φ_4 can be constructed similarly to φ_3.
- The first position (of the data word) and the first position after $atom(\#)$ have the same data value,

$$\varphi_5 = \exists x. \bigvee_{1 \le j \le n} (type(j) \wedge \tau(x) \wedge F(type(\#) \wedge X(type(\bar{j}) \wedge \tau(x))).$$

- The second position and the second position after $atom(\#)$ have the same data value,

$$\varphi_6 = \exists x. \bigvee_{\sigma \in \Sigma} (X(type(\sigma) \wedge \tau(x)) \wedge F(type(\#) \wedge XX(type(\overline{\sigma}) \wedge \tau(x))).$$

- The last occurrence of letters from $\{atom(1), \ldots, atom(n)\}$ and the last occurrence of letters $\{atom(\overline{1}), \ldots, atom(\overline{n})\}$ have the same data value,

$$\varphi_7 = \exists x.F \bigvee_{1 \le j \le n} \Big(type(j) \wedge \tau(x) \wedge X^{|u_j|+1} type(\#) \wedge F(type(\overline{j}) \wedge \tau(x) \wedge X^{|\overline{v_j}|}(\overline{X} false)) \Big).$$

- The last position and the last position before $atom(\#)$ have the same data value,

$$\varphi_8 = \exists x.F \bigvee_{\sigma \in \Sigma} \big(type(\sigma) \wedge \tau(x) \wedge X type(\#) \wedge F(type(\overline{\sigma}) \wedge \tau(x) \wedge \overline{X} false) \big).$$

- For every two consecutive occurrences of letters from $\{atom(1), \ldots, atom(n)\}$, there are two consecutive occurrences of letters from $\{atom(\overline{1}), \ldots, atom(\overline{n})\}$ with the same letters (by viewing $atom(j)$ the same as $atom(\overline{j})$) and the same data values,

$$\varphi_9 = G \bigwedge_{1 \le j_1, j_2 \le n} \Big((type(j_1) \wedge X^{|u_{j_1}|+1} type(j_2)) \rightarrow \exists x. \exists y. (\psi_1 \wedge F \psi_2) \Big),$$

where $\psi_1 = \tau(x) \wedge X^{|u_{j_1}|+1} \tau(y)$ and $\psi_2 = type(\overline{j_1}) \wedge \tau(x) \wedge X^{|\overline{v_{j_1}}|+1}(type(\overline{j}_2) \wedge \tau(y))$.
- For every two consecutive occurrences of letters from $\{atom(\sigma) \mid \sigma \in \Sigma\}$, there are two consecutive occurrences of letters from $\{atom(\overline{\sigma}) \mid \sigma \in \Sigma\}$ with the same letters (by viewing $atom(\sigma)$ the same as $atom(\overline{\sigma})$) and the same data values,

$$\varphi_{10} = G \bigwedge_{\sigma_1, \sigma_2 \in \Sigma} (\psi_0 \rightarrow \exists x. \exists y. (\psi_1 \wedge X \psi_2 \wedge F(\psi_3 \wedge X \psi_4))),$$

where $\psi_0 = type(\sigma_1) \wedge X(type(\sigma_2) \vee \vee_{1 \le j \le n}(type(j) \wedge X type(\sigma_2))))$, $\psi_1 = type(\sigma_1) \wedge \tau(x)$, $\psi_2 = (type(\sigma_2) \wedge \tau(y)) \vee \vee_{1 \le j \le n}(type(j) \wedge X(type(\sigma_2) \wedge \tau(y)))$, and $\psi_3 = type(\overline{\sigma_1}) \wedge \tau(x)$, $\psi_4 = (type(\overline{\sigma_2}) \wedge \tau(y)) \vee \vee_{1 \le j \le n}(type(\overline{j}) \wedge X(type(\overline{\sigma_2}) \wedge \tau(y)))$.

From the construction, we know that the instance of the PCP problem has a solution iff the $\exists^*$-VLTL formula $\varphi = \bigwedge_{1 \le i \le 10} \varphi_i$ is satisfiable. □

By a similar reduction from the nonemptiness of two-counter machines as in the proof of Theorem 4.1 of [13], we can show the following result.

Theorem 4.2. *The satisfiability and ω-satisfiability problems of $\forall$-VLTL$_{pnf}$ are undecidable.*

Proof. The proof is by a reduction from the nonemptiness problem of two-counter machines.

Let $\mathcal{A} = (Q, q_I, \delta, F)$ be a two-counter machine over the alphabet Σ, where q_I is the initial state, F is the set of final states, and $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \{inc_i, dec_i, ifz_i \mid i = 1, 2\} \times Q$.

The set of transition rules δ can be encoded by $\lceil \log(|\delta|) \rceil$ bits. Define AP as a set of atomic propositions of size $\lceil \log(|\delta|) \rceil$. In addition, let $T = \emptyset$ and $\mathbb{A} = \{a\}$. For each $(q, \sigma, \ell, q') \in \delta$, let $atom((q, \sigma, \ell, q'))$ denote the element of 2^{AP} corresponding to the binary encoding of (q, σ, ℓ, q'), and $type((q, \sigma, \ell, q'))$ denote the conjunction of atomic propositions or negated atomic propositions corresponding to the binary encoding of (q, σ, ℓ, q').

An accepting run of a two counter machine can be encoded by an $\mathbb{A}$-attributed data word w satisfying the following conditions,

1. $prj(w)$ is of the form

$$atom((q_0, \sigma_1, \ell_1, q_1)) atom((q_1, \sigma_2, \ell_2, q_3)) \ldots atom((q_{n-1}, \sigma_n, \ell_n, q_n))$$

   s.t. $q_0 = q_I$, for every $i : 1 \le i \le n$, $(q_{i-1}, \sigma_i, \ell_i, q_i) \in \delta$, and $q_n \in F$,
2. for every $j = 1, 2$, no two occurrences of $atom((q_{i-1}, \sigma_i, \ell_i, q_i))$ s.t. $\ell_i = inc_j$ (resp. $\ell_i = dec_j$) have the same data value,
3. for every $j = 1, 2$ and every occurrence of $atom((q_{i-1}, \sigma_i, \ell_i, q_i))$ s.t. $\ell_i = inc_j$, there is an occurrence of $atom((q_{i'-1}, \sigma_{i'}, \ell_{i'}, q_{i'}))$ s.t. $\ell_{i'} = dec_j$ on the right with the same data value, in addition, in between, there is no occurrence of $atom((q_{i''-1}, \sigma_{i''}, \ell_{i''}, q_{i''}))$ s.t. $\ell_{i''} = ifz_j$,
4. for every $j = 1, 2$ and every occurrence of $atom((q_{i-1}, \sigma_i, \ell_i, q_i))$ s.t. $\ell_i = dec_j$, there is an occurrence of $atom((q_{i'-1}, \sigma_{i'}, \ell_{i'}, q_{i'}))$ s.t. $\ell_{i'} = inc_j$ with the same data value.

The first condition is expressed by the following VLTL formula φ_1, without variables,

$$\varphi_1 = \psi_p \wedge \psi_i \wedge \psi_f \wedge \psi_t,$$

where ψ_p states that in each position of data words, at most one element of δ occurs,

$$\psi_p = \bigwedge_{\theta_1,\theta_2\in\delta,\theta_1\neq\theta_2} G\left(type(\theta_1) \to \overline{type(\theta_2)}\right),$$

ψ_i and ψ_f describes the initial and final state respectively,

$$\psi_i = \bigvee_{(q_I,\sigma,\ell,q)\in\delta} type((q_I,\sigma,\ell,q)),$$

$$\psi_f = F \bigvee_{(q,\sigma,\ell,q')\in\delta,q'\in F} \left(type((q,\sigma,\ell,q')) \wedge \overline{X} false\right),$$

and ψ_t states the conformance to the transition relation,

$$\psi_t = \bigwedge_{(q_1,\sigma_1,\ell_1,q_2)\in\delta} G\left(type((q_1,\sigma_1,\ell_1,q_2)) \to \overline{X} \bigvee_{(q_2,\sigma_2,\ell_2,q_3)\in\delta} type((q_2,\sigma_2,\ell_2,q_3))\right).$$

For $\ell = inc_j, dec_j$ (where $j = 1, 2$), let $\psi_{\ell,x} = x@a \wedge \bigvee_{(q,\sigma,\ell,q')\in\delta} type((q,\sigma,\ell,q'))$. In addition, for $\ell = ifz_1, ifz_2$, let $\psi_\ell = \bigvee_{(q,\sigma,\ell,q')\in\delta} type((q,\sigma,\ell,q'))$. Note that the formula $x@a$, instead of $\tau(x)$, is used in $\psi_{\ell,x}$.

The second condition is expressed by

$$\varphi_2 = \bigwedge_{j=1,2} \left(G\left(\psi_{inc_j,x} \to \overline{X} G(\overline{\psi_{inc_j,x}})\right) \wedge G\left(\psi_{dec_j,x} \to \overline{X} G(\overline{\psi_{dec_j,x}})\right)\right).$$

The third condition is expressed by

$$\varphi_3 = \bigwedge_{j=1,2} G\left(\psi_{inc_j,x} \to \left(XF\psi_{dec_j,x} \wedge \overline{X} G(\psi_{ifz_j} \to \overline{X} G\overline{\psi_{dec_j,x}})\right)\right).$$

The fourth condition is expressed by

$$\varphi_4 = \bigwedge_{j=1,2} \left(F\psi_{dec_j,x} \to F\psi_{inc_j,x}\right),$$

Then the formula $\varphi = \forall x.\ (\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4)$ defines the set of $\mathbb{A}$-attributed data words that encode accepting runs of $\mathcal{A}$.

From the construction, the language $\mathcal{L}(\mathcal{A})$ is nonempty iff φ is satisfiable. □

In [18], it is claimed that if $\mathbb{A}$ is a singleton, the satisfiability problem of the VLTL formulae in prenex normal form, where the quantifier prefixes are of the form $\exists^*\forall$, is decidable, which seems contradicting to Theorem 4.2. However, VLTL defined in [18] is incomparable with VLTL defined in this paper, in the sense that the atomic formulae $x@a$ are not available in [18] and the data variable comparison modalities are not available in VLTL defined in this paper. The fragment in [17], excluding the data variable comparison modalities, corresponds to RVLTL$_{pnf}$ in our framework (cf. Section 2.3). Moreover, in the conclusion of [18], it was mentioned that the satisfiability of "$\forall\forall$-RVLTL$_{pnf}$" is undecidable, by adapting the undecidability proof of the model checking problem of "$\exists\exists$-RVLTL$_{pnf}$" over variable Kripke structures in [17]. This statement is questionable since the definition of the logic in [18] is different from that in [17] (recall that the logic in [17] includes the modalities τ, while that in [18] does not). The fragment in [18], excluding the data variable comparison modalities, corresponds to RVLTL$^+_{pnf}$ in our framework. In the following, we clarify the decidability frontier of RVLTL$_{pnf}$ and RVLTL$^+_{pnf}$ and show that the two logics behave quite differently for the satisfiability problem.

RVLTL$^+_{pnf}$. The satisfiability of $\forall\forall$-RVLTL$^+_{pnf}$ is undecidable; moreover, if $\mathbb{A}$ is a singleton, then the satisfiability of $\forall$-RVLTL$^+_{pnf}$ is undecidable (cf. Theorem 4.3).

RVLTL$_{pnf}$. The satisfiability of $\forall\forall\exists$-RVLTL$_{pnf}$ and $\forall\exists\forall$-RVLTL$_{pnf}$ is undecidable; moreover, if $\mathbb{A}$ is a singleton, then the satisfiability of $\forall\exists$-RVLTL$_{pnf}$ is undecidable (cf. Theorem 4.5). On the other hand, the satisfiability of $\exists^*\forall^*$-RVLTL$_{pnf}$ is decidable (no matter whether $\mathbb{A}$ is a singleton or not, even extended with data variable comparison modalities), by utilizing a quantifier-elimination argument (cf. Section 4.1.4).

Theorem 4.3. *The satisfiability and ω-satisfiability problems of $\forall\forall$-RVLTL$^+_{pnf}$ are undecidable. In addition, if the set of attributes $\mathbb{A}$ is a singleton, then the satisfiability and ω-satisfiability problems of $\forall$-RVLTL$^+_{pnf}$ are undecidable.*

Proof. We first consider the situation that $\mathbb{A}$ is a singleton, say $\{a\}$. Then there is a unique data value at each position of $\mathbb{A}$-attributed data words.

The reduction is the same as that in the proof of Theorem 4.2, with the following modifications,

- $T = \{\tau\}$,
- the formula $x@a$ in $\psi_{\ell,x}$ is replaced by $\tau(x)$,
- moreover, a formula $\varphi_5 = G\ \tau$ is added to φ, that is, $\varphi = \forall x.\ \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5$.

The formula φ_5 guarantees that the parameterized atomic proposition τ occurs at each position, which removes the trivial satisfaction of φ by letting τ occur nowhere in the data words.

We then consider the situation that $\mathbb{A}$ is not a singleton. Then there are multiple data values at each position. The reduction in this case is an adaptation of the above reduction by setting $\varphi = \forall x \forall y.\ \bigwedge\limits_{0 \le i \le 5} \varphi_i$, where $\varphi_1, \ldots, \varphi_5$ are the same as above, and $\varphi_0 = F(\tau(x) \wedge \tau(y)) \to G(\tau(x) \leftrightarrow \tau(y))$. The formula φ_0 is used to avoid the bad situation that an occurrence of inc_j $(j = 1, 2)$ carries two distinct data values d_1, d_2, but there are two distinct occurrences of dec_j, with one carrying the data value d_1 and the other carrying the data value d_2. If such a situation happens, then the increments and decrements of the two-counter machine cannot be matched in the desired way and the validity of the zero tests cannot be guaranteed. $\square$

Remark 4.4. It is open whether the satisfiability and ω-satisfiability problems of $\forall$-RVLTL$^{+}_{pnf}$ are decidable, when $\mathbb{A}$ is not a singleton.

Theorem 4.5. *The satisfiability and ω-satisfiability problems of $\forall\forall\exists$-RVLTL$_{pnf}$ and $\forall\exists\forall$-RVLTL$_{pnf}$ are undecidable. In addition, if the set of attributes $\mathbb{A}$ is a singleton, then the satisfiability and ω-satisfiability problems of $\forall\exists$-RVLTL$_{pnf}$ are undecidable.*

Proof. We first consider the situation that $\mathbb{A}$ is a singleton, say $\{a\}$.

The reduction is the same as the case that $\mathbb{A}$ is a singleton in Theorem 4.3, with the following modifications: $\varphi_0 = \forall x \exists z.\ \bigwedge\limits_{1 \le i \le 5} \varphi_i$, where $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ are the same as in Theorem 4.3, and $\varphi_5 = G(\neg\tau(x) \to \tau(z))$. Since each position n of $\mathbb{A}$-attributed data words carries a bounded number of data values, there is $d \in \mathbb{D}$ such that $\neg\tau(x)$ is satisfied in the position n, so $\tau(z)$ is satisfied in the position n for some z. Therefore, φ_5 here plays the same role as $G\tau$ in Theorem 4.3.

We then consider the situation that $\mathbb{A}$ is not a singleton. The reduction is still the same as the case that $\mathbb{A}$ is not a singleton in Theorem 4.3, with the modification that $\varphi_0 = \forall x \forall y \exists z.\ \bigwedge\limits_{1 \le i \le 5} \varphi_i$ or $\varphi_0 = \forall x \exists z \forall y.\ \bigwedge\limits_{1 \le i \le 5} \varphi_i$, where $\varphi_1, \ldots, \varphi_4$ are the same as in Theorem 4.3, and $\varphi_5 = G(\neg\tau(x) \to \tau(z))$. $\square$

Remark 4.6. It is open whether the satisfiability and ω-satisfiability problems of $\forall\exists$-RVLTL$_{pnf}$ are decidable, when $\mathbb{A}$ is not a singleton.

Theorem 4.7. *The satisfiability and ω-satisfiability problems of $\exists\forall$-VLTL$^{noap}_{pnf}$, $\forall\exists$-VLTL$^{noap}_{pnf}$, and $\forall\forall$-VLTL$^{noap}_{pnf}$ are undecidable.*

Proof. We first consider $\exists\forall$-VLTL$^{noap}_{pnf}$.

From Theorem 4.2, we know that the satisfiability of $\forall$-VLTL$_{pnf}$ is undecidable. In the proof of Theorem 4.2, the atomic propositions from AP are essential to express the first condition, that is, the projection of a data word conforms to the transition relation of the two-counter machine. Since atomic propositions from AP are forbidden in $\exists\forall$-VLTL$^{noap}_{pnf}$, we propose a way to encode the atomic propositions by the equality relation of data values in the following.

Let $\varphi = \forall x.\psi$ be the formula constructed in the proof of the undecidability of $\forall$-VLTL$_{pnf}$ in Theorem 4.2. Suppose $\varphi = \forall x.\psi$ is in negation normal form. Our goal is to construct an $\exists\forall$-VLTL$^{noap}_{pnf}$ formula φ' such that φ is satisfiable iff φ' is satisfiable. The undecidability of the satisfiability of $\exists\forall$-VLTL$^{noap}_{pnf}$ then follows from Theorem 4.2. Note that the formula ψ uses $X, \overline{X}, F, G$ temporal operators, but not U or R, and it uses no parameterized atomic propositions from T. Without loss of generality, we assume that $T = \emptyset$.

Suppose $AP = \{p_1, \ldots, p_k\}$, $T = \emptyset$, and $\mathbb{A} = \{a\}$. Let $AP' = \emptyset$ and $T' = \{\tau'_0\}$. We intend to encode an $\mathbb{A}$-attributed data word w over $AP \cup T$ into an $\mathbb{A}$-attributed data word over $AP' \cup T'$ satisfying the following conditions,

- there is a data value d such that d occurs in all the positions $i(k+2)$ for $i \in \mathbb{N}$ (the position indices start from 0),
- τ'_0 occurs in all the positions $i(k+2)$ for $i \in \mathbb{N}$, but nowhere else,
- for each $i \in \mathbb{N}$, the position i of w is encoded by the block of w' from the position $i(k+2)$ to the position $(i+1)(k+2)-1$, where the data value d at the position i of w is put in the position $(i+1)(k+2)-1 = i(k+2)+(k+1)$ of w', and for each j, p_j (resp. τ_j) is true in the position i of w iff d occurs in the position $i(k+2)+j$ (resp. $i(k+2)+k+j$) of w'.

We construct a $VLTL^{noap}_{pnf}$ formula $\varphi' = \exists y \forall x.\ \psi_0 \wedge \psi'$ as follows.

- ψ_0 expresses that $\tau'_0(y)$ occurs in the position $i(k+2)$ for every $i \in \mathbb{N}$, but nowhere else,

$$\psi_0 = \tau'_0(y) \wedge G\left(\tau'_0(y) \rightarrow \left(\overline{X}^{(k+2)}\tau'_0(y) \wedge \bigwedge_{1 \le i < k+2} X^i \neg \tau'_0(y)\right)\right),$$

- ψ' is obtained from ψ by applying the following replacements in bottom-up along the syntax tree of ψ. Here we assume that ψ is in positive normal form (Note that some formulae in Theorem 4.2 are not in positive norm form, which are put on purpose to ease the reading).
  1. For each eventual occurrence of p_j (resp. $\neg p_j$) in ψ, replace p_j (resp. $\neg p_j$) with the formula $\tau'_0(y) \wedge X^j y@a$ (resp. $\tau'_0(y) \wedge X^j \neg y@a$), where $p_j \in AP$.
  2. For each persistent occurrence of p_j (resp. $\neg p_j$), replace p_j (resp. $\neg p_j$) with the formula $\tau'_0(y) \rightarrow X^j y@a$ (resp. $\tau'_0(y) \rightarrow X^j \neg y@a$), where $p_j \in AP$.
  3. For each eventual occurrence of $x@a$ (resp. $\neg x@a$), replace $x@a$ (resp. $\neg x@a$) with the formula $\tau'_0(y) \wedge X^{k+1} x@a$ (resp. $\tau'_0(y) \wedge X^{k+1} \neg x@a$).
  4. For each persistent occurrence of $x@a$ (resp. $\neg x@a$), replace $x@a$ (resp. $\neg x@a$) with the formula $\tau'_0(y) \rightarrow X^{k+1} x@a$ (resp. $\tau'_0(y) \rightarrow X^{k+1} \neg x@a$).
  5. For each eventual occurrence of the subformula of the form $X\phi$ (resp. $\overline{X}\phi$), replace $X\phi$ (resp. $\overline{X}\phi$) with the formula $\tau'_0(y) \wedge X^{k+2}(\tau'_0(y) \wedge \phi')$ (resp. $\tau'_0(y) \wedge \overline{X}^{k+2}(\tau'_0(y) \wedge \phi')$).
  6. For each persistent occurrence of the subformula of the form $X\phi$ (resp. $\overline{X}\phi$), replace $X\phi$ (resp. $\overline{X}\phi$) with the formula $\tau'_0(y) \rightarrow X^{k+2}(\tau'_0(y) \wedge \phi')$ (resp. $\tau'_0(y) \rightarrow \overline{X}^{k+2}(\tau'_0(y) \wedge \phi')$).

For instance, let $k = 2$ and $\varphi = XG((\neg p_1) \vee XFp_2)$, then the following formula φ' is constructed,

$$\begin{aligned}\varphi' \ = \ & \exists y.\ \tau'_0(y) \wedge X^4(\tau'_0(y) \wedge G[(\tau'_0(y) \rightarrow X \neg y@a) \vee \\ & (\tau'_0(y) \rightarrow X^4(\tau'_0(y) \wedge F(\tau'_0(y) \wedge X^2 y@a)))]).\end{aligned}$$

From this example, the reader may understand better why we need distinguish between eventual and persistent occurrences of the subformulae.

From the construction, we know that φ is satisfiable iff φ' is satisfiable.

For $\forall\exists$-VLTL$^{noap}_{pnf}$, the construction is similar, with the modification that φ' is replaced by $\varphi'' = \forall x \exists y.\ \psi_0 \wedge \psi'$. Because $\mathbb{A} = \{a\}$ is a singleton, it follows that φ is satisfiable iff φ'' is satisfiable.

For $\forall\forall$-VLTL$^{noap}_{pnf}$, let $\varphi''' = \forall y \forall x.\ (y@a \rightarrow (\psi_0 \wedge \psi'))$. Since there is exactly one data value occurring in the position 0 of an $\mathbb{A}$-attributed data word (recall that $\mathbb{A} = \{a\}$), it follows that φ is satisfiable iff φ''' is satisfiable.

Finally, it is easy to see that the proofs can be adapted to the ω-satisfiability of $\exists\forall$-VLTL$^{noap}_{pnf}$, $\forall\exists$-VLTL$^{noap}_{pnf}$ and $\forall\forall$-VLTL$^{noap}_{pnf}$. □

In the following, we state an undecidability result for the ω-satisfiability problem, while the corresponding satisfiability problem is decidable (cf. Theorem 4.10).

Theorem 4.8. *The ω-satisfiability problem of NN-$\exists^*$-VLTL is undecidable.*

Proof. We reduce from the nonemptiness of two-counter machines with incrementing errors over infinite words, which is known to be undecidable [11]. The reduction is similar to that in Theorem 4.2, the conjunction of the first three conditions there can be expressed by a NN-$\exists^*$-VLTL formula φ', with the formula ψ_f replaced by

$$\psi'_f = GF \bigvee_{(q,\sigma,\ell,q') \in \delta, q \in F} type((q,\sigma,\ell,q')).$$

Then the nonemptiness of a two-counter machine $\mathcal{A}$ with incrementing errors over infinite words is reduced to the ω-satisfiability of φ'. □

4.1.2. Decidability: NN-$\exists^*$-VLTL

We first present the encodings of $\mathbb{A}$-attributed data words into $\mathbb{A}'$-attributed data words where $\mathbb{A}'$ is a singleton.

Suppose $\mathbb{A} = \{a_0, \ldots, a_{K-1}\}$ and $\mathbb{A}' = \{a'\}$.

Suppose that $w = w_0 \ldots w_n$ is an $\mathbb{A}$-attributed data word over $AP \cup T$ s.t. for every $i : 1 \le i \le n$, $w_i = (A_i, ((B_{i,0}, d_{i,0}), \ldots, (B_{i,K-1}, d_{i,K-1})))$. Let $p' \notin AP \cup T$ and $AP' = AP \cup \{p'\}$. An $\mathbb{A}'$-*attributed encoding* of w, denoted by $enc(w)$, is a data word

$$AP = \{p\} \qquad T = \{\tau\} \qquad \mathbb{A} = \{a_0, a_1\}$$

$$w: \quad \begin{array}{c} \\ a_0 \\ a_1 \end{array} \begin{pmatrix} \emptyset \\ (\{\tau\}, d_1) \\ (\emptyset, d_2) \end{pmatrix} \begin{pmatrix} \{p\} \\ (\emptyset, d_3) \\ (\{\tau\}, d_4) \end{pmatrix} \begin{pmatrix} \{p\} \\ (\{\tau\}, d_5) \\ (\{\tau\}, d_6) \end{pmatrix}$$

$$enc(w):$$

$$(\{p'\}, (\{\tau\}, d_1))\ (\emptyset, (\emptyset, d_2))\ (\{p, p'\}, (\emptyset, d_3))\ (\{p\}, (\{\tau\}, d_4))\ (\{p, p'\}, (\{\tau\}, d_5))\ (\{p\}, (\{\tau\}, d_6))$$

Fig. 1. $\mathbb{A}'$-encoding of $\mathbb{A}$-attributed data words.

$w' = w'_{0,0} \dots w'_{0,K-1} \dots w'_{n,0} \dots w'_{n,K-1}$ over $AP' \cup T$ s.t. for every $i : 0 \le i \le n$, $w'_{i,0} = (A_i \cup \{p'\}, (B_{i,0}, d_{i,0}))$, and for every $j : 1 \le j \le K-1$, $w'_{i,j} = (A_i, (B_{i,j}, d_{i,j}))$. Fig. 1 shows an example of $\mathbb{A}'$-attributed encoding of $\mathbb{A}$-attributed data words, where $AP = \{p\}$, $T = \{\tau\}$, $K = 2$, $\mathbb{A} = \{a_0, a_1\}$, each position of w is encoded by two consecutive positions in $enc(w)$, and the atomic proposition p' holds in the positions of even indices, that is, $0, 2, \dots$.

We then present an encoding of VLTL formulae over $\mathbb{A}$-attributed data words to VLTL formulae over $\mathbb{A}'$-attributed data words.

Suppose that φ is a normalized VLTL formula. Then $enc(\varphi) = \varphi'_1 \wedge \varphi'_2$ with φ'_1 and φ'_2 defined as follows.

- φ'_1 puts restrictions on the occurrences of p' and the atomic propositions from AP,

$$\varphi'_1 = p' \wedge G\left[p' \to \begin{array}{c} \bigwedge\limits_{p \in AP} [(\wedge_{0 \le i \le K-1} X^i p) \vee (\wedge_{0 \le i \le K-1} X^i \neg p)] \wedge \\ \bigwedge\limits_{1 \le i \le K-1} X^i \neg p' \wedge \overline{X}^K p' \end{array} \right].$$

  Intuitively, φ'_1 states that p' occurs in the first position, for every occurrence of p' in some position, p' will occur in the K-th position after it if there is such a position, but does not occur in between, moreover, for every $p \in AP$, either p occurs in all the positions between two adjacent occurrences of p', or occurs in none of them.
- φ'_2 is obtained from φ by the following procedure.
  1. Replace every eventual occurrence of $X\phi$ (resp. $\overline{X}\phi$) by $p' \wedge X^K \phi$ (resp. $p' \wedge \overline{X}^K \phi$).
  2. Replace every persistent occurrence of $X\phi$ (resp. $\overline{X}\phi$) by $p' \to X^K \phi$ (resp. $p' \to \overline{X}^K \phi$).
  3. For every $p \in AP$, replace every eventual occurrence of p (resp. $\neg p$) by $p' \wedge \vee_{0 \le i \le K-1} X^i p$ (resp. $p' \wedge \vee_{0 \le i \le K-1} X^i \neg p$), and every persistent occurrence of p (resp. $\neg p$) by $p' \to \vee_{0 \le i \le K-1} X^i p$ (resp. $p' \to \vee_{0 \le i \le K-1} X^i \neg p$). The formula $\vee_{0 \le i \le K-1} X^i p$ (resp. $\vee_{0 \le i \le K-1} X^i \neg p$) states that p (resp. $\neg p$) occurs in one of the next K positions. This is sound since for every $p \in AP$, φ'_1 requires that either p occurs in all of them or none of them.
  4. For every $\tau \in T$ and $x \in Var$, replace every eventual occurrence of $\tau(x)$ (resp. $\neg\tau(x)$) by $p' \wedge \vee_{0 \le i \le K-1} X^i \tau(x)$ (resp. $p' \wedge \vee_{0 \le i \le K-1} X^i \neg\tau(x)$), and every persistent occurrence of $\tau(x)$ (resp. $\neg\tau(x)$) by $p' \to \vee_{0 \le i \le K-1} X^i \tau(x)$ (resp. $p' \to \vee_{0 \le i \le K-1} X^i \neg\tau(x)$).
  5. For every $i \in [K]$, replace every eventual occurrence of $x@a_i$ (resp. $\neg x@a_i$) with $p' \wedge X^i x@a'$ (resp. $p' \wedge X^i \neg x@a'$), every persistent occurrence of $x@a_i$ (resp. $\neg x@a_i$) with $p' \to X^i x@a'$ (resp. $p' \to X^i \neg x@a'$).

Proposition 4.9. *For every VLTL formula φ over $\mathbb{A}$-attributed data words, it holds that $enc(\mathcal{L}(\varphi)) = \mathcal{L}(enc(\varphi))$.*

Theorem 4.10. *The satisfiability problem of NN-$\exists^*$-VLTL is decidable and non-primitive recursive.*

Proof. The proof is by a reduction to the nonemptiness problem of AWRA.

Let φ be a NN-$\exists^*$-VLTL sentence. From the definition of $enc(\varphi)$, it is not hard to observe that $enc(\varphi)$ is also a NN-$\exists^*$-VLTL sentence. In addition, from Proposition 4.9, we know that $enc(\mathcal{L}(\varphi)) = \mathcal{L}(enc(\varphi))$. Therefore, it is sufficient to consider the satisfiability of $enc(\varphi)$ over $\mathbb{A}'$-attributed data words (recall that $\mathbb{A}'$ is a singleton).

Since the quantifiers are not nested, without loss of generality, we assume that there is only one variable, say x, used in φ. Note that the variable x may be reused and existentially quantified for many times.

Our goal is to construct an AWRA $\mathcal{A}_{enc(\varphi)}$ s.t. $\mathcal{L}(\mathcal{A}_{enc(\varphi)}) = \mathcal{L}(enc(\varphi))$. We will construct the AWRA by an induction on the syntax of NN-$\exists^*$-VLTL formulae.

The construction of an AWRA for atomic formulae or negated atomic formulae, Boolean operators and temporal operators is similar to the construction of alternating automata from LTL formulae (cf. [1]).

For existential quantification $\varphi = \exists x.\psi$, suppose that an AWRA $\mathcal{A}_\psi$ with the initial state q_ψ has been constructed, then an AWRA $\mathcal{A}_\varphi$ can be constructed by adding a state q_φ as the new initial state and adding the transitions $\delta(q_\varphi) = guess(q_\psi)$.

The non-primitive recursive lower bound is by a reduction from the nonemptiness problem of two-counter machines with incrementing errors over finite words (cf. [11]). The reduction is similar to that in Theorem 4.2, where all the four conditions, except the last, can be expressed in NN-$\exists^*$-VLTL. □

4.1.3. Decidability: $\forall$-VLTL$^{gdap}_{pnf}$

In the following, we state and prove the decidability result for $\forall$-VLTL$^{gdap}_{pnf}$.

Theorem 4.11. *The satisfiability and ω-satisfiability problems of $\forall$-VLTL$^{gdap}_{pnf}$ are decidable.*

Proof. Suppose $\varphi = \forall x.\ \psi$ is a $\forall$-VLTL$^{gdap}_{pnf}$ sentence over $AP \cup T$.

From the definition of $enc(\cdot)$, we know that $enc(\varphi) = \varphi'_1 \wedge \varphi'_2$ and $\varphi'_2 = \forall x.\ \psi'$ for some quantifier free VLTL formula ψ'. Then $enc(\varphi)$ can be rewritten into $\forall x.\ (\varphi'_1 \wedge \psi')$, since no variables occur in φ'_1. So $enc(\varphi)$ is a $\forall$-VLTL$_{pnf}$ sentence over $AP' \cup T$, where $AP' = AP \cup \{p'\}$. The formula φ'_1 requires that p' occurs in the positions pos s.t. $pos \equiv 0 \bmod K$. It is not hard to observe that if φ is a $\forall$-VLTL$^{gdap}_{pnf}$ sentence, then ψ' can be rewritten into a quantifier free VLTL formula where all the occurrences of p and $\neg p$ for $p \in AP$ are guarded by the positive occurrences of x (that is, $\tau(x)$ for some $\tau \in T$, or $x@a'$). For instance,

- for an eventual occurrence of $p \wedge \tau(x)$ in ψ s.t. $p \in AP$ and $\tau \in T$, it is transformed into $(p' \wedge \bigvee\limits_{0\le i\le K-1} X^i p) \wedge (p' \wedge \bigvee\limits_{0\le i\le K-1} X^i \tau(x))$ in ψ', which is equivalent to $p' \wedge \bigvee\limits_{0\le i\le K-1} X^i(p \wedge \tau(x))$, since either none of $X^i p$ holds or all of them hold,
- for an eventual occurrence of $\neg(p \wedge \tau(x))$ in ψ s.t. $p \in AP$ and $\tau \in T$, it is equivalent to $\neg p \vee \neg\tau(x) \equiv (\neg p \wedge \tau(x)) \vee \neg\tau(x)$, which is then transformed into $[p' \wedge \bigvee\limits_{0\le i\le K-1} X^i(\neg p \wedge \tau(x))] \vee [p' \wedge \bigwedge\limits_{0\le i\le K-1} X^i \neg\tau(x)]$ in ψ',
- for an eventual occurrence of $p \wedge x@a_i$ in ψ s.t. $p \in AP$ and $a_i \in \mathbb{A}$, it is transformed into $(p' \wedge \bigvee\limits_{0\le j\le K-1} X^j p) \wedge (p' \wedge X^i x@a')$ in ψ', which can be replaced by $p' \wedge X^i(p \wedge x@a')$, since either none of $X^j p$ holds or all of them hold,
- for an eventual occurrence of $\neg(p \wedge x@a_i)$ in ψ s.t. $p \in AP$ and $a_i \in \mathbb{A}$, it is equivalent to $(\neg p \wedge x@a_i) \vee \neg x@a_i$, which is then transformed into $(p' \wedge X^i(\neg p \wedge x@a')) \vee (p' \wedge X^i \neg x@a')$ in ψ'.

By abuse of notation, we still denote the resulting formula by ψ'. Note that the formula $\forall x.\ (\varphi'_1 \wedge \psi')$ is not a $\forall$-VLTL$^{gdap}_{pnf}$ formula since the occurrences of p' are not guarded.

To continue the proof, we introduce the following notation. Suppose $w = w_0 \dots w_n$ is an $\mathbb{A}'$-attributed data word over $AP' \cup T$. Then $prj_{AP'\cup T}(w) = w_0|_{AP'\cup T} \dots w_n|_{AP'\cup T}$, where for every $i : 0 \le i \le n$, suppose that $w_i = (A_i, (B_i, d_i))$, then $w_i|_{AP'\cup T} = (A_i, B_i)$. The definition of $prj_{AP'\cup T}(\cdot)$ can be naturally generalized to languages of $\mathbb{A}'$-attributed data words.

From Proposition 4.9, we know that the satisfiability of φ over $\mathbb{A}$-attributed data words is reduced to the nonemptiness of the language $\mathcal{L}(enc(\varphi))$ over $\mathbb{A}'$-attributed data words. The nonemptiness of $\mathcal{L}(enc(\varphi))$ is then reduced to the nonemptiness of $prj_{AP'\cup T}(\mathcal{L}(enc(\varphi)))$.

In the following, we will construct an EDA $\mathcal{D}_{enc(\varphi)}$ from $enc(\varphi) = \forall x.\ (\varphi'_1 \wedge \psi')$ s.t. $\mathcal{L}(\mathcal{D}_{enc(\varphi)}) = prj_{AP'\cup T}(\mathcal{L}(enc(\varphi)))$. The decidability then follows from Theorem 2.10.

The EDA $\mathcal{D}_{enc(\varphi)} = (AP' \cup T, \mathcal{A}, \mathcal{B})$ is constructed as follows.

- $\mathcal{A}$ is the identity transducer which checks that the atomic propositions from AP' occur in a desired way, that is, p' occurs exactly in the positions pos s.t. $pos \equiv 0 \bmod K$, and for each $p \in AP$ and a position $pos : pos \equiv 0 \bmod K$, either p occurs in all the positions $pos, pos+1, \dots, pos+K-1$, or p occurs in none of them.
- $\mathcal{B}$ is constructed from $\forall x.\ (\varphi'_1 \wedge \psi')$ by the following procedure.
  1. Construct an LTL formula ψ'' from ψ' as follows. Intuitively, ψ'' verifies that the constraint ψ' is satisfied for a fixed valuation of x (the positions where the data value assigned to x occurs are exactly the positions where the symbols $(A, B, 1) \in 2^{AP'} \times 2^T \times \{1\}$ occur).
     - Replace every occurrence of p' (resp. $\neg p'$) by the formula $(\{p'\}, 0) \vee \bigvee\limits_{p'\in A\subseteq AP', B\subseteq T} (A, B, 1)$ (resp. $(\emptyset, 0) \vee \bigvee\limits_{p'\notin A\subseteq AP', B\subseteq T} (A, B, 1)$).
     - Replace every occurrence of $p \wedge \tau(x)$ (resp. $\neg(p \wedge \tau(x))$) by the formula $\bigvee\limits_{p\in A\subseteq AP', \tau\in B\subseteq T} (A, B, 1)$ (resp. $(\{p'\}, 0) \vee (\emptyset, 0) \vee \bigvee\limits_{p\notin A\subseteq AP', B\subseteq T} (A, B, 1) \vee \bigvee\limits_{A\subseteq AP', \tau\notin B\subseteq T} (A, B, 1)$). Similarly for $\neg p \wedge \tau(x)$ and $\neg(\neg p \wedge \tau(x))$.

- Replace every occurrence of $p \wedge x@a'$ (resp. $\neg(p \wedge x@a')$) by the formula $\bigvee\limits_{p \in A \subseteq AP', B \subseteq T} (A, B, 1)$ (resp. $(\{p'\}, 0) \vee (\emptyset, 0) \vee \bigvee\limits_{p \notin A \subseteq AP', B \subseteq T} (A, B, 1)$).
- Replace every occurrence of $\tau(x)$ by the formula $\bigvee\limits_{A \subseteq AP', \tau \in B \subseteq T} (A, B, 1)$.
- Replace every occurrence of $\neg\tau(x)$ by the formula $(\{p'\}, 0) \vee (\emptyset, 0) \vee \bigvee\limits_{A \subseteq AP', \tau \notin B \subseteq T} (A, B, 1)$.
- Replace every occurrence of $x@a'$ by the formula $\bigvee\limits_{A \subseteq AP', B \subseteq T} (A, B, 1)$.
- Replace every occurrence of $\neg x@a'$ by the formula $(\{p'\}, 0) \vee (\emptyset, 0)$.

2. Construct a finite state automaton $\mathcal{A}_{\psi''}$ over the alphabet $\Sigma' = 2^{AP'} \times 2^T \times \{1\} \cup 2^{\{p'\}} \times \{0\}$ from ψ''.
3. Finally, from $\mathcal{A}_{\psi''} = (\Sigma', Q, Q_0, \delta, Q_f)$, construct a finite automaton $\mathcal{B} = (\Sigma, Q \times [K], Q_0 \times \{0\}, \delta', Q_f \times \{K-1\})$. Intuitively, $\mathcal{B}$ guesses the occurrences of p' in the positions pos s.t. $pos \equiv 0 \bmod K$, and simulates $\mathcal{A}_{\psi''}$. Formally, in $\mathcal{B}$, $\Sigma = (2^{AP'} \times 2^T \times \{1\}) \cup \{0\}$, and δ' is defined by the following rules,
   - if $(q, (A, B, 1), q') \in \delta$ for $A \subseteq AP'$ s.t. $p' \in A$, then $((q, 0), (A \cap AP, B, 1), (q', 1)) \in \delta'$,
   - if $(q, (A, B, 1), q') \in \delta$ for $A \subseteq AP'$ s.t. $p' \notin A$, then for each $j : 0 < j < K$, $((q, j), (A, B, 1), (q', j+1 \bmod K)) \in \delta'$,
   - if $(q, (\{p'\}, 0), q') \in \delta$, then $((q, 0), 0, (q', 1)) \in \delta'$,
   - if $(q, (\emptyset, 0), q') \in \delta$, then for each $j : 0 < j < K$, $((q, j), 0, (q', j+1 \bmod K)) \in \delta'$.

For the ω-satisfiability problem, the construction of the ω-EDA is adapted from the EDA constructed above, with the following modifications.

- Construct from ψ'' a Büchi automaton $\mathcal{A}_{\psi''}$ over the alphabet $\Sigma' = 2^{AP'} \times 2^T \times \{1\} \cup 2^{\{p'\}} \times \{0\}$.
- From $\mathcal{A}_{\psi''} = (\Sigma', Q, Q_0, \delta, Q_f)$, a Büchi automaton $\mathcal{B} = (\Sigma, Q \times [K], Q_0 \times \{0\}, \delta', Q_f \times [K])$ is constructed, with Σ and δ' defined the same as above.

The decidability of the ω-satisfiability of $\forall$-VLTL$^{gdap}_{pnf}$ then follows from Theorem 2.10. □

4.1.4. *Decidability: $\exists^*\forall^*$-RVLTL$_{pnf}$*

Theorem 4.12. *The satisfiability of $\exists^*\forall^*$-RVLTL$_{pnf}$ is in EXPSPACE and PSPACE-hard. In particular, the satisfiability of $\exists^k\forall^*$-RVLTL$_{pnf}$ (where k is a constant) is PSPACE-complete. The results hold even for the extension of $\exists^*\forall^*$-RVLTL$_{pnf}$ with data variable comparison modalities.*

Proof. The proof is obtained by a special way to eliminate the universal variable quantifiers from the formulae.

Without loss of generality, we assume that for each $\exists^*\forall^*$-RVLTL$_{pnf}$ sentence φ, no variables in φ are quantified twice.

Let $\varphi = \exists x_1 \ldots \exists x_k \forall y_1 \ldots \forall y_l.\ \psi$ be an $\exists^*\forall^*$-RVLTL$_{pnf}$ sentence. For each function f from $\{1, \ldots, l\}$ to $\{0, 1, \ldots, k\}$, define the formula $elm_f(\psi)$ as follows: For each $i \in \{1, \ldots, l\}$,

- if $f(i) = 0$, then for each $\tau \in T$, replace each occurrence of $\tau(y_i)$ (resp. $\neg\tau(y_i)$) with *false* (resp. *true*),
- otherwise, replace each occurrence of y_i with $x_{f(i)}$.

Note that the formulae $elm_f(\psi)$ contain only the variables $x_1, \ldots, x_k$. In addition, let $elm_\forall(\varphi)$ denote the sentence $\exists x_1 \ldots \exists x_k.\ \wedge_f\ elm_f(\psi)$. The size of $elm_\forall(\varphi)$ is exponential over k. In the following, we will show that φ is satisfiable iff $elm_\forall(\varphi)$ is satisfiable. The complexity upper bounds then follow from the fact that the satisfiability of $\exists^*$-RVLTL$_{pnf}$ is PSPACE-complete (cf. Proposition 2.5). The complexity lower bound is from that of LTL.

Claim. φ is satisfiable iff $elm_\forall(\varphi)$ is satisfiable.

Proof of the Claim. The "Only if" direction is easy. We only present the proof for the "If" direction.

Suppose $elm_\forall(\varphi)$ is satisfiable. Then there is an $\mathbb{A}$-attributed data word w such that $w \models \exists x_1 \ldots \exists x_k.\ \wedge_f\ elm_f(\psi)$. So there are $d_1, \ldots, d_k$ such that $w \models_\lambda \wedge_f\ elm_f(\psi)$, where λ is the function that assigns d_j to x_j for each $j : 1 \le j \le k$. Let w' be the data word obtained from w as follows: For each occurrence of (β, d) such that $\beta \in 2^T$ and $d \notin \{d_1, \ldots, d_k\}$, replace (β, d) with $(\emptyset, d)$. In the following, we show that $w' \models_\lambda \forall y_1 \ldots \forall y_l.\psi$. Thus $w' \models \exists x_1 \ldots \exists x_k \forall y_1 \ldots \forall y_l.\psi$ and φ is satisfiable.

To show that $w' \models_\lambda \forall y_1 \ldots \forall y_l.\psi$, it is sufficient to show that for each tuple of data values $d'_1, \ldots, d'_l$, $w' \models_{\lambda[d'_1/y_1, \ldots, d'_l/y_l]} \psi$.

Let $d'_1, \ldots, d'_l$ be a tuple of data values and f be the function from $\{1, \ldots, l\}$ to $\{0, 1, \ldots, k\}$ such that for each $i : 1 \le i \le l$, $f(i) = 0$ if $d'_i \notin \{d_1, \ldots, d_k\}$, and $f(i) = \min(\{j \mid 1 \le j \le k, d'_i = d_j\})$ otherwise. Then $w \models_\lambda elm_f(\psi)$. Let d' be a data value not occurring in w and λ' be the function that extends λ by assigning d' to y_i if $f(i) = 0$, and $d_{f(i)}$ to y_i otherwise. Because $w \models_\lambda elm_f(\psi)$, and for each $i : 1 \le i \le l$ such that $f(i) = 0$, $\tau(y_i)$ (resp. $\neg\tau(y_i)$) is replaced by *false* (resp. *true*) when constructing $elm_f(\psi)$ from ψ, moreover, $\lambda'(\tau(y_i))$ (resp. $\lambda'(\neg\tau(y_i))$) evaluates to *false* (resp. *true*) in each position of w, we deduce that $w \models_{\lambda'} \psi$.

For each $\tau \in T$ and $i : 1 \le i \le l$ such that $f(i) = 0$, we know that $\lambda'(y_i) = d'$. Since d' does not occur in w, according the construction of w' from w, we know that d' does not occur in w'. Therefore, $\lambda'(\tau(y_i))$ (resp. $\lambda'(\neg\tau(y_i))$) evaluates to *false* (resp. *true*) in each position of w'. Because $w \models_{\lambda'} \psi$ and w' and w are identical in the positions that hold the data values $d_1, \ldots, d_k$, we deduce that $w' \models_{\lambda'} \psi$.

Moreover, for each $\tau(y_i)$ such that $f(i) = 0$, since $d'_i \notin \{d_1, \ldots, d_k\}$, we deduce that $(\lambda[d'_1/y_1, \ldots, d'_l/y_l])(\tau(y_i))$ (resp. $(\lambda[d'_1/y_1, \ldots, d'_l/y_l])(\neg\tau(y_i))$) evaluates to *false* (resp. *true*) in each position of w'. Because λ' and $\lambda[d'_1/y_1, \ldots, d'_l/y_l]$ agree on the data values assigned to $x_1, \ldots, x_k$ as well as to the variables y_i such that $f(i) \neq 0$, we conclude that $w' \models_{\lambda[d'_1/y_1,\ldots,d'_l/y_l]} \psi$. $\square$

The results of Theorem 4.12 can be extended to the situation that the data variable comparison modalities e.g. $x = y$ and $\neg x = y$ are available by adapting the construction of $elm_\forall(\varphi)$ slightly to include the equality and inequality information between the variables y_i's s.t. $f(i) = 0$. Thus Theorem 4.12 extends the results in [18], where the satisfiability of $\exists^*\forall$-RVLTL$_{pnf}$ (with data variable comparisons) was claimed to be decidable. On the other hand, it is open whether the ω-satisfiability of $\exists^*\forall^*$-RVLTL$_{pnf}$ is decidable.

4.2. Model checking problem

In this section, we prove the undecidability of the model checking and ω-model checking problems for fragments of VLTL. We will only present the proofs for the model checking problem and the proofs can be easily extended to the ω-model checking problem.

Since a VKS can be defined to accept the set of all $\mathbb{A}$-attributed data words where $\mathbb{A}$ is a singleton, we deduce the following result from Theorem 4.1.

Corollary 4.13. *The model checking and ω-model checking problems of $\forall^*$-VLTL are undecidable.*

Proof. We prove the corollary by a reduction from the satisfiability problem of $\exists^*$-VLTL over $\mathbb{A}$-attributed data words where $\mathbb{A}$ is a singleton.

Suppose φ is an $\exists^*$-VLTL sentence over $AP \cup T$. Then the negation of φ, more precisely, $\overline{\varphi}$, is a $\forall^*$-VLTL sentence.

Define a VKS $\mathcal{K} = (AP, X, S, R, S_0, I, L, L')$ as follows:

- $X = \{x\}$,
- $S = S_0 = \{s_i \mid 1 \le i \le |2^{AP} \cup 2^T|\}$,
- $R = \{(s, s') \mid \forall s, s' \in S\}$,
- $I(s) = \{true\}$ for all $s \in S$,
- L is a bijection function from S to $2^{AP} \cup 2^T \times X$,
- $L'(e) = \{reset\} \times X$ for every $e \in R$.

It is easy to see that $\mathcal{L}(\mathcal{K})$ is the set of all the $\mathbb{A}$-attributed data words over $AP \cup T$ where $\mathbb{A}$ is a singleton. Thus, φ is satisfiable iff $\overline{\varphi}$ is not valid iff $\mathcal{L}(\mathcal{K}) \nsubseteq \mathcal{L}(\overline{\varphi})$ iff $\mathcal{K} \not\models \overline{\varphi}$. $\square$

Similarly, we deduce from Theorem 4.2 and Theorem 4.7 the following two results.

Corollary 4.14. *The model checking and ω-model checking problems of $\exists$-VLTL$_{pnf}$ are undecidable.*

Corollary 4.15. *The model checking and ω-model checking problems of $\forall\exists$-VLTL$^{noap}_{pnf}$, $\exists\forall$-VLTL$^{noap}_{pnf}$, and $\exists\exists$-VLTL$^{noap}_{pnf}$ are undecidable.*

Theorem 4.16. *The model checking and ω-model checking problems of $\exists$-RVLTL$_{pnf}$ are undecidable.*

Proof. In the following, we only present the arguments for model checking problem. It is easy to see that the arguments can be extended to ω-model checking problem.

We prove the theorem by a reduction from the satisfiability problem of $\forall$-VLTL$_{pnf}$. Let φ' be the sentence obtained from φ constructed in the proof of Theorem 4.2 by replacing *x@a* with $\tau(x)$, where τ is a newly introduced parameterized atomic proposition. Let $\mathcal{A}$ be the two counter machine defined as in the proof of Theorem 4.2. It is easy to see that $\mathcal{L}(\mathcal{A})$ is nonempty iff φ' is satisfiable over the $\mathbb{A}$-attributed data words in which τ occurs at each position (Note that $\mathbb{A} = \{a\}$). Note that φ' is a $\forall$-RVLTL$_{pnf}$ sentence and $\overline{\varphi'}$ (the negation of φ') is an $\exists$-RVLTL$_{pnf}$ sentence.

Define a VKS $\mathcal{K} = (AP, X, S, R, S_0, I, L, L')$ as follows:

- $X = \{x\}$,

- $S = S_0 = \{s_i \mid 1 \le i \le |2^{AP}|\}$,
- $R = \{(s, s') \mid \forall s, s' \in S\}$,
- $I(s) = \{true\}$ for all $s \in S$,
- L is a bijection function from S to $\{P \cup \{(\tau, x)\} \mid P \subseteq AP\}$,
- $L'(e) = \{(reset, x)\}$ for every $e \in R$.

It is easy to see that $\mathcal{L}(\mathcal{K})$ is the set of all the X-attributed data words over $AP \cup T$ in which τ occurs in each position (Note that X is a singleton). Thus, φ' is satisfiable over X-attributed data words in which τ occurs in each position iff $\mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\varphi') \neq \emptyset$ iff $\mathcal{L}(\mathcal{K}) \nsubseteq \mathcal{L}(\overline{\varphi'})$ iff $\mathcal{K} \not\models \overline{\varphi'}$.

Therefore, the model checking problem of $\exists$-RVLTL$_{pnf}$ is undecidable. □

We can also deduce from Theorem 4.8 the following result.

Corollary 4.17. *The ω-model checking problem of NN-$\forall^*$-VLTL is undecidable.*

In the following, we will deduce from the decidability of the satisfiability problem of NN-$\exists^*$-VLTL the decidability of the model checking problem of the dual fragment. For this purpose, we show the following preliminary result.

Proposition 4.18. *Let $\mathcal{K}$ be a VKS. Then $enc(\mathcal{L}(\mathcal{K}))$ can be defined by an AWRA.*

Proof. Let $\mathcal{K} = (AP \cup T, X, S, R, S_0, I, L, L')$ be a VKS and $K = |X|$. Suppose $AP = \{p_1, \dots, p_l\}$, $T = \{\tau_1, \dots, \tau_m\}$, $X = \{x_0, \dots, x_{K-1}\}$, and $S = \{s_1, \dots, s_n\}$. Without loss of generality, we assume that $S_0 = \{s_1, \dots, s_r\}$ for some $r : 1 \le r \le n$. In addition, we assume that for every $s \in S$, a linear order is fixed for $R(s)$, that is, the set of successors of s. Let k be the maximum number of successors of states in $\mathcal{K}$.

Let $p' \notin AP \cup T$, $AP' = AP \cup \{p'\}$, and $\mathbb{A}' = \{a'\}$. Then $enc(\mathcal{L}(\mathcal{K}))$ is a set of $\mathbb{A}'$-attributed data words over $AP' \cup T$.

Construct an AWRA $\mathcal{A} = (AP' \cup T, Q, q_0, \delta)$ to define $enc(\mathcal{L}(\mathcal{K}))$ as follows. Intuitively, when reading an $\mathbb{A}'$-attributed data word w', $\mathcal{A}$ nondeterministically chooses a finite path π of $\mathcal{K}$ and verifies that w' conforms to the invariants $I(s)$ for $s \in S$ and the constraints induced by the edge-labeling function L' (e.g. for an edge $(s, s') \in R$ s.t. $(reset, x) \notin L'(s, s')$, the value of x in s is the same as that in s'). Note that during the construction, for the convenience of the presentation, we use the arbitrary positive Boolean combinations of states in the definition of δ. Auxiliary states can be introduced to transform δ into the standard definition. More specifically, $\mathcal{A}$ is constructed as follows.

- Q includes all the states occurring in the definition of δ, where
  - the states $((s, s'), j)$ for $j \in [K]$ are used to record the choice of the edges out of s (where the indices j are added due to the fact that each position of w corresponds to K consecutive positions in $enc(w)$),
  - the states $(s, -, j)$ where $j \in [K]$ are used to indicate that the current path of $\mathcal{K}$ stops in the state s,
  - the states $(s, 0, \psi)$, the states $(s, j, x_i \ op \ x_{i'})$, and the states $(s, j, (x_i)_{op})$ are used to enforce the invariants $I(s)$ for $s \in S$, where ψ is a subformula of $I(s)$, $j \in [K]$ and $op \in \{=, \neq\}$,
  - the states $(s, 0, \psi)$, the states (s, j, p_i) and $(s, j, \neg p_i)$, and the states $(s, j', \tau_i(x_j))$ and $(s, j', \neg\tau_i(x_j))$ are used to verify the conformance to the state-labeling function L, where ψ is a subformula of $\varphi_{L(s)}$ (defined later), $p_i \in AP$, $\tau_i \in T$, $x_j \in X$ and $j, j' \in [K-1]$,
  - the states $((s, s'), j, x_i)$ and the states $((s, s'), j, (x_i)_=)$ are used to enforce the constraints between the pair of variable valuations in s and s' respectively induced by the edge-labeling function $L'(s, s')$.
- q_0 is the initial state.
- δ is defined as follows.

  For each state $s \in S$, let $\varphi_{L(s)}$ denote the formula $\big(\theta_1 \wedge (\theta_2 \wedge \dots (\theta_{l-1} \wedge \theta_l) \dots\big) \wedge (\eta_{1,0} \wedge (\eta_{1,1} \wedge \dots (\eta_{1,K-1} \wedge (\eta_{2,0} \wedge (\cdots \wedge (\eta_{m,K-2} \wedge \eta_{m,K-1}) \dots))) \dots))$, where for every $i : 1 \le i \le l$, $\theta_i = p_i$ if $p_i \in L(s)$, $\theta_i = \neg p_i$ otherwise; for every $i : 1 \le i \le m$ and $j : 0 \le j \le K-1$, $\eta_{i,j} = \tau_i(x_j)$ if $(\tau_i, x_j) \in L(s)$, $\eta_{i,j} = \neg\tau_i(x_j)$ otherwise.
  - $\delta(q_0) = (s_1, 0) \vee (s_2, 0) \vee \cdots \vee (s_r, 0)$.
  - For every $s \in S$, let $R(s) = \{s'_0, \dots, s'_i\}$, then $\delta((s, 0)) = p' \wedge (s, 0, I(s)) \wedge (s, 0, \varphi_{L(s)}) \wedge ([(s, s'_1, 0) \vee \cdots \vee (s, s'_i, 0)] \vee (s, -, 0))$. Intuitively, when the current path is in the state s, either some successor $s'_{i'}$ of s (where $0 \le i' \le i$) is chosen as the next state in the path and the state of $\mathcal{A}$ is changed to $((s, s'_{i'}), 0)$, or $(s, -, 0)$ is chosen and the path stops in the state s.
  - For every $s \in S$ and $j \in [K-1]$, $\delta((s, -, j)) = \nabla_0(s, -, j+1)$, moreover, $\delta((s, -, K-1)) = \overline{\nabla_0}?$.
  - For every $s \in S$ and every subformula $\psi = \psi_1 \ op \ \psi_2$ of $I(s)$ (where $op = \vee, \wedge$), $\delta((s, 0, \psi)) = (s, 0, \psi_1) \ op \ (s, 0, \psi_2)$.
  - For every $s \in S$, every subformula $x_i \ op \ x_{i'}$ (where $op \in \{=, \neq\}$ and $i < i'$) of $I(s)$, and every $j : 0 \le j < i$, $\delta((s, j, x_i \ op \ x_{i'})) = \nabla_0(s, j+1, x_i \ op \ x_{i'})$, $\delta((s, i, x_i \ op \ x_{i'})) = store((s, i, (x_{i'})_{op}))$.
  - For every $s \in S$, $i : 0 \le i \le K-1$, $j : 0 \le j < i$, and $op \in \{=, \neq\}$, $\delta((s, j, (x_i)_{op})) = \nabla_0(s, j+1, (x_i)_{op})$, $\delta((s, i, (x_i)_=)) = eq$, $\delta((s, i, (x_i)_{\neq})) = \overline{eq}$.
  - For every $s \in S$ and every subformula $\psi = \psi_1 \wedge \psi_2$ of $\varphi_{L(s)}$, $\delta((s, 0, \psi)) = (s, 0, \psi_1) \wedge (s, 0, \psi_2)$,

– For every $s \in S$, $i : 1 \leq i \leq l$, $\delta((s, 0, p_i)) = p_i$, $\delta((s, 0, \neg p_i)) = \neg p_i$.
– For every $s \in S$, $i : 1 \leq i \leq m$, $j : 0 \leq j \leq K - 1$ and $j' : 0 \leq j' < j$, $\delta((s, j', \tau_i(x_j))) = \triangledown_0(s, j' + 1, \tau_i(x_j))$, $\delta((s, j', \neg\tau_i(x_j))) = \triangledown_0(s, j' + 1, \neg\tau_i(x_j))$, moreover, $\delta((s, j, \tau_i(x_j))) = \tau_i$, and $\delta((s, j, \neg\tau_i(x_j))) = \neg\tau_i$.
– For every $(s, s') \in R$, let $Y = \{x_i \mid (reset, x_i) \notin L'(s, s')\}$, then $\delta(((s, s'), 0)) = \triangledown_0((s, s'), 1) \wedge \bigwedge_{x_i \in Y} ((s, s'), 0, x_i)$. Intuitively, $((s, s'), 0, x_i)$ is used to verify that the value of x_i in the state s' is the same as that of s.
– For every $s \in S$, let $R(s) = \{s'_0, \dots, s'_i\}$, then for every $i : 0 \leq i' \leq i$ and $j : 0 \leq j < K - 1$, $\delta(((s, s'_{i'}), j)) = \triangledown_0((s, s'_{i'}), j + 1)$, $\delta(((s, s'_{i'}), K - 1)) = \triangledown_0(s'_{i'}, 0)$.
– For every $(s, s') \in R$, $i : 0 \leq i \leq K - 1$ and $j : 0 \leq j < i$, $\delta(((s, s'), j, x_i)) = \triangledown_0((s, s'), j + 1, x_i)$, $\delta(((s, s'), i, x_i)) = store(((s, s'), i, (x_i)_=))$. Intuitively, $((s, s'), i, (x_i)_=)$ means that the data value assigned to x_i on s has been stored in the register, and this value should be equal to that of s'.
– For every $(s, s') \in R$, $i : 0 \leq i \leq K - 1$ and $j : i \leq j < K - 1$, $\delta(((s, s'), j, (x_i)_=)) = \triangledown_0((s, s'), j + 1, (x_i)_=)$.
– For every $s \in S$, let $R(s) = \{s'_0, \dots, s'_i\}$, then for every $i' : 0 \leq i' \leq i$ and $j : 0 \leq j \leq K - 1$, $\delta(((s, s'_{i'}), K - 1, (x_j)_=)) = \triangledown_0(s'_{i'}, 0, (x_j)_=)$. Intuitively, by going to the state $(s'_{i'}, 0, (x_j)_=)$, the fact that the data value stored in the register, which is the data value of x_j in the state s, should be equal to the value of x_j in the state $s'_{i'}$, will be checked. □

Corollary 4.19. *The model checking problem of NN-∀*-VLTL is decidable and non-primitive recursive.*

Proof. For every VKS $\mathcal{K} = (AP \cup T, X, S, R, S_0, I, L, L')$ and every NN-∀*-VLTL sentence φ over X-attributed data words, we have $\mathcal{K} \not\models \varphi$ iff $\mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\overline{\varphi}) \neq \emptyset$ iff $enc(\mathcal{L}(\mathcal{K})) \cap enc(\mathcal{L}(\overline{\varphi})) \neq \emptyset$ iff $enc(\mathcal{L}(\mathcal{K})) \cap \mathcal{L}(enc(\overline{\varphi})) \neq \emptyset$.

From the proof of Theorem 4.10, we know that an AWRA $\mathcal{A}_{enc(\overline{\varphi})}$ can be constructed from $enc(\overline{\varphi})$ s.t. $\mathcal{L}(\mathcal{A}_{enc(\overline{\varphi})}) = \mathcal{L}(enc(\overline{\varphi}))$.

From Proposition 4.18, we know that an AWRA $\mathcal{A}_{\mathcal{K}}$ can be constructed from $\mathcal{K}$ s.t. $\mathcal{L}(\mathcal{A}_{\mathcal{K}}) = enc(\mathcal{L}(\mathcal{K}))$.

Because the language defined by AWRAs are closed under intersection, it follows that an AWRA can be constructed to define $\mathcal{L}(\mathcal{A}_{\mathcal{K}}) \cap \mathcal{L}(\mathcal{A}_{enc(\overline{\varphi})})$. The decidability of the model checking problem then follows from Theorem 2.8.

For the lower bound, it follows from Theorem 4.10 and the following argument: Since a VKS $\mathcal{K}$ can be constructed to define the set of all $\mathbb{A}$-attributed data words where $\mathbb{A}$ is a singleton, we have that for every NN-∃*-VLTL formula φ, φ is satisfiable over $\mathbb{A}$-attributed data words iff $\mathcal{K} \not\models \overline{\varphi}$. □

Then we deduce from the satisfiability and ω-satisfiability problems of ∀-VLTL$^{gdap}_{pnf}$ the decidability of the model checking and ω-model checking problems of the dual fragment.

Corollary 4.20. *The model checking and ω-model checking problems of ∃-VLTL$^{gdap}_{pnf}$ are decidable.*

Proof. We first consider model checking problem.

Let $\mathcal{K} = (AP \cup T, X, S, R, S_0, I, L, L')$ be a VKS and φ be an ∃-VLTL$^{gdap}_{pnf}$ sentence over X-attributed data words. From Proposition 4.9, $\mathcal{K} \models \varphi$ iff $\mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\overline{\varphi}) = \emptyset$ iff $enc(\mathcal{L}(\mathcal{K})) \cap \mathcal{L}(enc(\overline{\varphi})) = \emptyset$.

On the other hand, it is not hard to observe that $enc(\mathcal{L}(\mathcal{K})) \cap \mathcal{L}(enc(\overline{\varphi})) \neq \emptyset$ iff $prj_{AP' \cup T}(enc(\mathcal{L}(\mathcal{K}))) \cap prj_{AP' \cup T}(\mathcal{L}(enc(\overline{\varphi}))) \neq \emptyset$. The "only if" direction is trivial. For the "if" direction, suppose $prj_{AP' \cup T}(enc(\mathcal{L}(\mathcal{K}))) \cap prj_{AP' \cup T}(\mathcal{L}(enc(\overline{\varphi}))) \neq \emptyset$. Then there are $w' \in enc(\mathcal{L}(\mathcal{K}))$ and $w'' \in \mathcal{L}(enc(\overline{\varphi}))$ s.t. $prj_{AP' \cup T}(w') = prj_{AP' \cup T}(w'')$. Since in w' and w'', p' occurs in the same positions, it follows that $w' = w''$. Therefore, $w' \in enc(\mathcal{L}(\mathcal{K})) \cap \mathcal{L}(enc(\overline{\varphi}))$.

From the proof of Theorem 4.11, we know that from $prj_{AP' \cup T}(enc(\overline{\varphi}))$, an equivalent EDA can be constructed. From Theorem 2.10, it is sufficient to construct an EDA defining $prj_{AP' \cup T}(enc(\mathcal{L}(\mathcal{K})))$.

It is not hard to observe that $prj_{AP' \cup T}(enc(\mathcal{L}(\mathcal{K})))$ can be defined by a nondeterministic register automaton (NRA) (cf. [38]). On the other hand, it is known that from a NRA, an equivalent DA can be constructed (cf. [56]). Since a DA is a special EDA, it follows that an EDA can be constructed to define $prj_{AP' \cup T}(enc(\mathcal{L}(\mathcal{K})))$.

The argument for the ω-model checking problem is similar, with NRAs and EDAs replaced by ω-NRAs and ω-EDAs. □

5. Decision problems of VCTL

5.1. Undecidability

By adding a universal path quantifier A before every temporal operator of φ in the proof of Theorem 4.1, we get a reduction to the satisfiability problem of ∃*-AVCTL.

Corollary 5.1. *The satisfiability and ω-satisfiability problems of ∃*-AVCTL formulae are undecidable.*

Proof. The reduction in Theorem 4.1 can be adapted into a reduction to $\exists^*$-AVCTL as follows: We first normalize the formula φ in Theorem 4.1 by replacing every subformula $\psi_1 \to \psi_2$ with $\overline{\psi_1} \vee \psi_2$. Then we add the universal path quantifier A before every occurrence of temporal operators. Let φ' be the resulting $\exists^*$-AVCTL formula.[5]

In the following, we will show that φ is satisfiable over data words iff φ' is satisfiable over data trees.

Suppose φ is satisfiable, then there is a data word w s.t. $w \models \varphi$. Let t_w be a k-ary data tree where the data word on every path of t_w is w, then it is not hard to see that $t_w \models \varphi'$.

On the other hand, suppose that φ' is satisfiable. Then there are $k \geq 1$ and a k-ary data tree t s.t. $t \models \varphi'$. Let φ_i for $i = 1, \ldots, 10$ be the subformulae defined as in the proof of Theorem 4.1. Take an arbitrary path π in t, we want to show that for every formula φ_i (where $i = 1, \ldots, 10$), we have $w_\pi \models \varphi_i$.

We use φ_{10} to exemplify the proof.

Let φ'_{10} be the $\exists^*$-AVCTL formula by adding A before every occurrence of temporal operators in φ_{10}.

Then $t \models \varphi'_{10}$. This implies that for every $\sigma_1, \sigma_2 \in \Sigma$ and every node π_i on π, $t|_{\pi_i} \models \psi'_0 \to \exists x \exists y(\psi'_1 \wedge AX\psi'_2 \wedge AF(\psi'_3 \wedge AX\psi'_4))$, where ψ'_0 is the formula obtained from ψ_0 by adding the existential path quantifier E before every occurrence of temporal operators, $\psi'_1, \psi'_2, \psi'_3$ are the formulae obtained from respectively ψ_1, ψ_2, ψ_3 by adding A before every occurrence of temporal operators.

To show $w_\pi \models \varphi_{10}$, that is for all $\sigma_1, \sigma_2 \in \Sigma$, $w_\pi \models \psi_{\sigma_1,\sigma_2}$, it is sufficient to show that for every $\sigma_1, \sigma_2 \in \Sigma$ and $i : 0 \leq i < |\pi|$, if $(w_\pi)^i \models \psi_0$, then $(w_\pi)^i \models \exists x \exists y(\psi_1 \wedge X\psi_2 \wedge F(\psi_3 \wedge X\psi_4))$.

Suppose $(w_\pi)^i \models \psi_0$, then $t|_{\pi_i} \models \psi'_0$. From the fact that $t|_{\pi_i} \models \psi'_0 \to \exists x \exists y(\psi'_1 \wedge AX\psi'_2 \wedge AF(\psi'_3 \wedge AX\psi'_4))$, we know that $t|_{\pi_i} \models \exists x \exists y(\psi'_1 \wedge AX\psi'_2 \wedge AF(\psi'_3 \wedge AX\psi'_4))$. Therefore, there is an assignment $\lambda : \{x, y\} \to D$ s.t. $t|_{\pi_i} \models_\lambda \psi'_1 \wedge AX\psi'_2 \wedge AF(\psi'_3 \wedge AX\psi'_4)$. Since $(w_\pi)^i$ is a data word corresponding to a path in $t|_{\pi_i}$, it follows that $(w_\pi)^i \models_\lambda \psi_1 \wedge X\psi_2 \wedge F(\psi_3 \wedge X\psi_4)$. From this, we conclude that $(w_\pi)^i \models \exists x \exists y(\psi_1 \wedge X\psi_2 \wedge F(\psi_3 \wedge X\psi_4))$. □

The argument in the proof of Corollary 5.1 relies essentially on the universal path quantifiers A. Later on, we show that the satisfiability is decidable if only existential path quantifiers E are allowed, no matter whatever variable quantifications are used. In addition, unlike VLTL, from the undecidability of the satisfiability problem of a fragment of *VCTL*, the undecidability of the model checking problem of the dual fragment does not follow directly. The reason is that there does not exist a variable Kripke structure which defines the set of all $\mathbb{A}$-attributed data trees or even the set of all k-ary $\mathbb{A}$-attributed data trees for a fixed k. For instance, later on, we will show that the satisfiability problem of $\exists^*$-*EVCTL* (in fact *EVCTL*) is decidable, while the model checking problem for $\forall^*$-AVCTL is undecidable.

Similarly, from Theorem 4.2 and Theorem 4.7, we deduce the following result.

Corollary 5.2. *The satisfiability and ω-satisfiability problems of $\forall$-AVCTL$_{pnf}$ are undecidable.*

Corollary 5.3. *The satisfiability and ω-satisfiability problems of $\exists\forall$-VCTL$^{noap}_{pnf}$, $\forall\exists$-VCTL$^{noap}_{pnf}$ and $\forall\forall$-VCTL$^{noap}_{pnf}$ are undecidable.*

Moreover, from Theorem 4.8, we deduce the following result.

Corollary 5.4. *The ω-satisfiability problem of NN-$\exists^*$-VCTL is undecidable.*

Next we consider the model checking and ω-model checking problems.

Theorem 5.5. *The model checking and ω-model checking problems are undecidable for the following fragments: $\forall^*$-AVCTL, $\forall^*$-EVCTL, $\exists\exists$-VCTL$^{noap}_{pnf}$, $\forall\exists$-VCTL$^{noap}_{pnf}$, $\exists\forall$-VCTL$^{noap}_{pnf}$, NN-$\exists^*$-VCTL. Moreover, the ω-model checking problem of NN-$\forall^*$-VCTL is undecidable.*

Proof. We present the arguments for the model checking problem. The arguments can be easily extended to the ω-model checking problem.

We prove the theorem by reductions from the satisfiability problems of $\exists^*$-VLTL and $\forall$-VLTL over $\mathbb{A}$-attributed data words where $\mathbb{A}$ is a singleton.

We first show the argument for the model checking problem of $\forall^*$-AVCTL.

Let φ be an $\exists^*$-VLTL sentence over $AP \cup T$. We will construct a VKS $\mathcal{K}$ and an $\exists^*$-EVCTL sentence φ' s.t. φ is satisfiable iff $\mathcal{K} \not\models \overline{\varphi'}$. Note that $\overline{\varphi'}$ is a $\forall^*$-AVCTL sentence.

The idea of the reduction is as follows: We construct a VKS $\mathcal{K}$ which is a single loop (without branchings). Thus each computation tree of $\mathcal{K}$ is in fact a data word. Then we obtain from φ by adding existential path quantifiers E before every temporal operator occurring in φ (plus some other modifications) to obtain φ'. Since $\mathcal{K}$ is a linear structure, the satisfaction of φ' over the computation trees of $\mathcal{K}$ mimics the satisfaction of φ over data words.

[5] If φ is not normalized, then for every subformula $\psi_1 \to \psi_2$, E should be added before each occurrence of temporal operators in ψ_1, so that we still get an $\exists^*$-AVCTL formula.

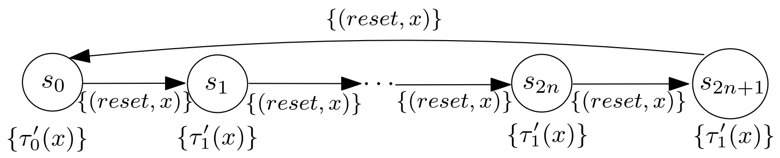


Fig. 2. The Kripke structure.

Suppose $AP = \{p_1, \ldots, p_m\}$, $T = \{\tau_1, \ldots, \tau_{n-m}\}$ (where $m \leq n$), and $\tau'_0, \tau'_1 \notin AP \cup T$. Define the VKS $\mathcal{K} = (AP' \cup T', \{x\}, S, R, S_0, I, L, L')$ as follows: $AP' = \emptyset$, $T' = \{\tau'_0, \tau'_1\}$, $S = \{s_0, s_1, \ldots, s_{2n+1}\}$, $R = \{(s_i, s_{i+1 \bmod 2n+2}) \mid 0 \leq i \leq 2n+1\}$, $S_0 = \{s_0\}$, for every $s_i \in S$, $I(s_i) = true$, $L(s_0) = \{(\tau'_0, x)\}$, $L(s_i) = \{(\tau'_1, x)\}$ for every $i : 1 \leq i \leq 2n+1$, and $L'(s_i, s_{i+1 \bmod 2n+2}) = \{(reset, x)\}$ for every $i : 0 \leq i \leq 2n+1$ (see Fig. 2).

Notice that in $\mathcal{K}$, T' only contains two parameterized atomic propositions. The set of atomic propositions AP will be encoded by equalities and inequalities between the data values of two adjacent τ'_1-labeled positions in $\mathcal{K}$. Thus each position (A, B, d) in an $\mathbb{A}$-attributed data word over $AP \cup T$ will be encoded by a segment of computation traces in $\mathcal{K}$ of length $2n+2$ s.t. the position 0 is labeled by τ'_0, the position $2i-1$ and $2i$ encode the satisfaction on $A \cup B$ of the i-th atomic proposition from $AP \cup T$, and the last position in the segment holds the data value d. In addition, x is reset on each edge $(s_i, s_{i+1 \bmod 2n+2})$ $(0 \leq i \leq 2n+1)$ so that an arbitrary data value can be assigned to x on each position $s_0, s_1, \ldots, s_{2n+1}$.

Construct the $\exists^*$-EVCTL sentence $\varphi' ::= \exists y.\ (\tau'_0(y) \wedge \varphi'_0 \wedge \varphi'_1)$ as follows.

- φ'_0 restricts the format of the computation traces of $\mathcal{K}$,

$$\varphi'_0 = EG[\tau'_0(y) \rightarrow ((E\overline{X})^{2n+2}\tau'_0(y) \wedge \wedge_{1 \leq i \leq n}(EX)^{2i-1}\tau'_1(y))].$$

  Intuitively, φ'_0 states that if (τ'_0, d_0) (assume that the data value d_0 is assigned to y) occurs in some position, then (τ'_0, d_0) will occur in the $(2n+2)$-th position after it if there is such a position, and (τ'_1, d_0) will occur in all the $(2i-1)$-th positions for $1 \leq i \leq n$ after it (But not necessarily occur in the $2i$-th position).
- φ'_1 is constructed from φ by the following procedure.
  1. For every eventual occurrence of $X\phi$ (resp. $\overline{X}\phi$), replace $X\phi$ (resp. $\overline{X}\phi$) by $\tau'_0(y) \wedge X^{2n+2}\phi$ (resp. $\tau'_0(y) \wedge \overline{X}^{2n+2}\phi$),
  2. for every persistent occurrence of $X\phi$ (resp. $\overline{X}\phi$), replace $X\phi$ (resp. $\overline{X}\phi$) by $\tau'_0(y) \rightarrow X^{2n+2}\phi$ (resp. $\tau'_0(y) \rightarrow \overline{X}^{2n+2}\phi$),
  3. for every proposition $p_i \in AP$, replace every eventual occurrence of p_i (resp. $\neg p_i$) by $\tau'_0(y) \wedge X^{2i-1}(\tau'_1(y) \wedge X\tau'_1(y))$ (resp. $\tau'_0(y) \wedge X^{2i-1}(\tau'_1(y) \wedge X\neg\tau'_1(y))$), and replace every persistent occurrence of p_i (resp. $\neg p_i$) by $\tau'_0(y) \rightarrow X^{2i-1}(\tau'_1(y) \wedge X\tau'_1(y))$ (resp. $\tau'_0(y) \rightarrow X^{2i-1}(\tau'_1(y) \wedge X\neg\tau'_1(y))$),
  4. for every proposition $\tau_i \in T$ and $x \in Var$, replace every eventual occurrence of $\tau_i(x)$ (resp. $\neg\tau_i(x)$) by $\tau'_0(y) \wedge X^{2(m+i)-1}(\tau'_1(y) \wedge X\tau'_1(y)) \wedge X^{2n+1}\tau'_1(x)$ (resp. $\tau'_0(y) \wedge [X^{2(m+i)-1}(\tau'_1(y) \wedge X\neg\tau'_1(y)) \vee X^{2n+1}\neg\tau'_1(x)]$), and replace every persistent occurrence of $\tau_i(x)$ (resp. $\neg\tau_i(x)$) by $\tau'_0(y) \rightarrow [X^{2(m+i)-1}(\tau'_1(y) \wedge X\tau'_1(y)) \wedge X^{2n+1}\tau'_1(x)]$ (resp. $\tau'_0(y) \rightarrow [X^{2(m+i)-1}(\tau'_1(y) \wedge X\neg\tau'_1(y)) \vee X^{2n+1}\neg\tau'_1(x)]$),
  5. add E before every occurrence of temporal operators.

For instance, suppose $AP = \emptyset$ and $T = \{\tau_1, \tau_2\}$, then the $\exists^*$-EVCTL formula corresponding to $\exists^*$-VLTL formula $\exists x.\ G(\neg\tau_1(x) \vee FX\tau_2(x))$ is $\exists y.[\tau'_0(y) \wedge \varphi'_0 \wedge \exists x.EG(\psi_1 \vee EF(\tau'_0(y) \wedge (EX)^6\psi_2))]$, where $\psi_1 = \tau'_0(y) \rightarrow [EX(\tau'_1(y) \wedge EX\neg\tau'_1(y))) \vee (EX)^5\neg\tau'_1(x)]$ and $\psi_2 = \tau'_0(y) \wedge (EX)^3(\tau'_1(y) \wedge EX\tau'_1(y)) \wedge (EX)^5\tau'_1(x)$.

Then from the construction, we know that φ is satisfiable iff there is a computation tree t of $\mathcal{K}$ s.t. $t \models \varphi'$, that is, iff $\mathcal{K} \not\models \overline{\varphi'}$.

Because all the computation trees of $\mathcal{K}$ are just computation traces, the same reduction works for $\forall^*$-EVCTL, by replacing A with E.

Next we consider the model checking problem of $\exists\exists$-VCTL$^{noap}_{pnf}$.

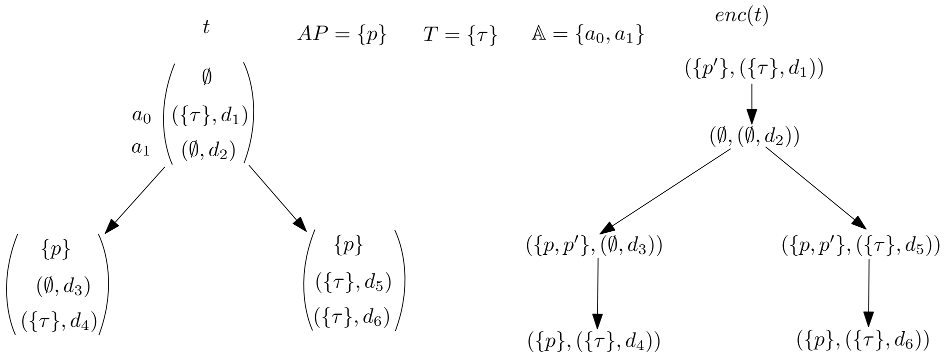
We reduce from the satisfiability problem of $\forall$-VLTL$_{pnf}$ over $\mathbb{A}$-attributed data words where $\mathbb{A}$ is a singleton.

Let $\varphi = \forall x.\ \psi$ be a normalized $\forall$-VLTL$_{pnf}$ sentence.

We construct a VKS $\mathcal{K}$ and a $\forall\forall$-VCTL$^{noap}_{pnf}$ formula φ' s.t. φ is satisfiable iff $\mathcal{K} \not\models \overline{\varphi'}$. Note that $\overline{\varphi'}$ is an $\exists\exists$-VCTL$^{noap}_{pnf}$ sentence.

The construction of the VKS $\mathcal{K}$ is the same as above. The formula φ' is constructed as $\forall x \forall y.[(\varphi'_0 \wedge \tau'_0(y)) \rightarrow \varphi'_1]$, where φ'_0 is the same as above and φ'_1 is obtained from ψ by doing the same replacements as in the construction of φ'_1 from φ above.

The argument for the construction is as follows: $\forall x.\psi$ is satisfiable iff there is a computation tree t of $\mathcal{K}$ s.t. $t \models \forall y.((\varphi'_0 \wedge \tau'_0(y)) \rightarrow \forall x.\varphi'_1)$, i.e. $t \models \forall x \forall y.((\varphi'_0 \wedge \tau'_0(y)) \rightarrow \varphi'_1)$. This is equivalent to $\mathcal{K} \not\models \exists x \exists y.(\varphi'_0 \wedge \tau'_0(y) \wedge \overline{\varphi'_1})$.



t
$AP = \{p\}$
$T = \{\tau\}$
$\mathbb{A} = \{a_0, a_1\}$
$enc(t)$
$\emptyset$
a_0 $(\{\tau\}, d_1)$
a_1 $(\emptyset, d_2)$
$\{p\}$ $(\emptyset, d_3)$ $(\{\tau\}, d_4)$
$\{p\}$ $(\{\tau\}, d_5)$ $(\{\tau\}, d_6)$
$(\{p'\}, (\{\tau\}, d_1))$
$(\emptyset, (\emptyset, d_2))$
$(\{p, p'\}, (\emptyset, d_3))$
$(\{p, p'\}, (\{\tau\}, d_5))$
$(\{p\}, (\{\tau\}, d_4))$
$(\{p\}, (\{\tau\}, d_6))$

Fig. 3. $\mathbb{A}'$-attributed encoding of $\mathbb{A}$-attributed data trees.

For the model checking problem of $\exists\forall$-VCTL$^{noap}_{pnf}$ (resp. $\forall\exists$-VCTL$^{noap}_{pnf}$), the reduction is the same as $\exists\exists$-VCTL$^{noap}_{pnf}$, with the formula φ' replaced by $\forall x \exists y.(\varphi'_0 \wedge \tau'_0(y) \wedge \varphi'_1)$ (resp. $\exists y \forall x.(\varphi'_0 \wedge \tau'_0(y) \wedge \varphi'_1)$).

Then, we consider the model checking problem of NN-$\exists^*$-VCTL. We still reduce from the satisfiability of $\forall$-VLTL$_{pnf}$ over $\mathbb{A}$-attributed data words where $\mathbb{A}$ is a singleton.

Let $\forall x.\ \psi$ be a $\forall$-VLTL$_{pnf}$ sentence.

To avoid nesting quantifiers, we add one atomic proposition $\{p'_0\}$ to the VKS $\mathcal{K}$. More specifically, $\mathcal{K}$ is obtained by adapting the construction above as follows: $AP' = \{p'_0\}$, $T' = \{\tau'\}$, $L(s_0) = \{p'_0, (\tau', x)\}$, and $L(s_i) = \{(\tau', x)\}$ for every $i : 1 \le i \le 2n+1$.

Construct the NN-$\forall^*$-VCTL formula φ' as $\forall x.\ \varphi'_1$, where φ'_1 is constructed from ψ by the following procedure.

1. For every eventual occurrence of $X\phi$ (resp. $\overline{X}\phi$), replace $X\phi$ (resp. $\overline{X}\phi$) by $p'_0 \wedge X^{2n+2}\phi$ (resp. $p'_0 \wedge \overline{X}^{2n+2}\phi$).
2. For every persistent occurrence of $X\phi$ (resp. $\overline{X}\phi$), replace $X\phi$ (resp. $\overline{X}\phi$) by $p'_0 \rightarrow X^{2n+2}\phi$ (resp. $p'_0 \rightarrow \overline{X}^{2n+2}\phi$).
3. For every proposition $p_i \in AP$, replace every eventual occurrence of p_i (resp. $\neg p_i$) by $p'_0 \wedge X^{2i-1}\forall y.(\tau'(y) \rightarrow X\tau'(y))$ (resp. $p'_0 \wedge X^{2i-1}\forall y.(\tau'(y) \rightarrow X\neg\tau'(y))$), and replace every persistent occurrence of p_i (resp. $\neg p_i$) by $p'_0 \rightarrow X^{2i-1}\forall y.(\tau'(y) \rightarrow X\tau'(y))$ (resp. $p'_0 \rightarrow X^{2i-1}\forall y.(\tau'(y) \rightarrow X\neg\tau'(y))$).
4. For every proposition $\tau_i \in T$ and $x \in Var$, replace every eventual occurrence of $\tau_i(x)$ (resp. $\neg\tau_i(x)$) by $p'_0 \wedge X^{2(m+i)-1}\forall y.(\tau'(y) \rightarrow X\tau'(y)) \wedge X^{2n+1}\tau'(x)$ (resp. $p'_0 \wedge [(X^{2(m+i)-1}\forall y.(\tau'(y) \rightarrow X\neg\tau'(y))) \vee X^{2n+1}\neg\tau'(x)]$), and replace every persistent occurrence of $\tau_i(x)$ (resp. $\neg\tau_i(x)$) by $p'_0 \rightarrow [X^{2(m+i)-1}\forall y.(\tau'(y) \rightarrow X\tau'(y)) \wedge X^{2n+1}\tau'(x)]$ (resp. $p'_0 \rightarrow [(X^{2(m+i)-1}\forall y.(\tau'(y) \rightarrow X\neg\tau'(y))) \vee X^{2n+1}\neg\tau'(x)]$).
5. Add E before every occurrence of temporal operators.

From the construction, we know that $\forall x.\psi$ is satisfiable iff there is a computation tree t of $\mathcal{K}$ s.t. $t \models \varphi'$ iff $\mathcal{K} \not\models \overline{\varphi'}$.

Finally, let us consider the ω-model checking problem of NN-$\forall^*$-VCTL. Let $\mathcal{K}$ be the VKS constructed as in the model checking problem of NN-$\exists^*$-VCTL above. Let φ' be the NN-$\exists^*$-VLTL formula constructed in the proof of Theorem 4.8. From φ', we construct a formula φ'' just as the construction of φ'_1 from ψ above, with the adaption that the formulae $\forall y.(\tau'(y) \rightarrow X\tau'(y))$ are replaced by $\exists y.\tau'(y) \wedge X\tau'(y)$.

It is easy to see that φ'' is still an NN-$\exists^*$-VCTL formula and $\overline{\varphi''}$ is an NN-$\forall^*$-VCTL formula.

Then $\mathcal{K} \not\models_\omega \overline{\varphi''}$ iff there is $t \in \mathcal{T}_\omega(\mathcal{K})$ s.t. $t \models \varphi''$ iff φ' is ω-satisfiable. From Theorem 4.8, we conclude that the ω-model checking problem of NN-$\forall^*$-VCTL is undecidable. $\square$

5.2. Decidability

This section is devoted to the decidability results of the decision problems of VCTL formulae.

5.2.1. Non-nested existential data variable quantifiers

Theorem 5.6. *The satisfiability problem of NN-$\exists^*$-VCTL is decidable and non-primitive recursive.*

Before we give the proof of Theorem 5.6, similar to the proof for NN-$\exists^*$-VLTL, we first define $\mathbb{A}'$-attributed encodings of $\mathbb{A}$-attributed data trees and $enc(\varphi)$ for NN-$\exists^*$-VCTL formulae φ.

Similar to the $\mathbb{A}'$-encodings of $\mathbb{A}$-attributed data words, we define $\mathbb{A}'$-attributed encodings of $\mathbb{A}$-attributed data trees as follows (cf. Fig. 3 for an example). Let $\mathbb{A} = \{a_0, \ldots, a_{K-1}\}$ and $\mathbb{A}' = \{a'\}$. Suppose that $t = (Z, L)$ is a k-ary $\mathbb{A}$-attributed data tree over $AP \cup T$ s.t. for every $z \in Z$, $L(z) = (A_z, ((B_{z,0}, d_{z,0}), \ldots, (B_{z,K-1}, d_{z,K-1})))$. Let $p' \notin AP \cup T$ and $AP' = AP \cup \{p'\}$. An *$\mathbb{A}'$-attributed encoding* of t, denoted by $enc(t)$, is a data tree $t' = (Z', L')$ over $AP' \cup T$ s.t. Z' is a k-ary tree satisfying the following conditions,

- for every $z = i_1 \dots i_n \in [k]^*$, we have $i_1 \dots i_n \in Z$ iff $0^{K-1} i_1 \dots 0^{K-1} i_n 0^{K-1} \in Z'$,
- for every $z = i_1 \dots i_n \in Z$, $L'(0^{K-1} i_1 \dots 0^{K-1} i_n) = (A_z \cup \{p'\}, (B_{z,0}, d_{z,0}))$, and for every $j : 1 \leq j \leq K-1$, $L'(0^{K-1} i_1 \dots 0^{K-1} i_n 0^j) = (A_z, (B_{z,j}, d_{z,j}))$.

Proposition 5.7. *Let $\mathcal{K}$ be a VKS. Then $enc(\mathcal{T}(\mathcal{K}))$ can be defined by an ATRA.*

Proof. Let $\mathcal{K} = (AP \cup T, X, S, R, S_0, I, L, L')$ be a VKS and $K = |X|$.

Suppose $AP = \{p_1, \dots, p_l\}$, $T = \{\tau_1, \dots, \tau_m\}$, $X = \{x_0, \dots, x_{K-1}\}$, and $S = \{s_1, \dots, s_n\}$. Without loss of generality, we assume that $S_0 = \{s_1, \dots, s_r\}$ for some $r : 1 \leq r \leq n$. In addition, we assume for every $s \in S$, a linear order is fixed for $R(s)$, that is, the set of successors of s. Let k be the maximum number of successors of states in $\mathcal{K}$.

Let $p' \notin AP \cup T$ and $AP' = AP \cup \{p'\}$.

Construct an ATRA $\mathcal{A} = (AP' \cup T, Q, q_0, \delta)$ over k-ary $\mathbb{A}'$-attributed data trees as follows. Intuitively, when reading an $\mathbb{A}'$-attributed data tree t', $\mathcal{A}$ check that t' is indeed a data tree obtained by unwinding $\mathcal{K}$ as follows,

- $\mathcal{A}$ nondeterministically chooses to stop in the current state s, or to continue unwinding by creating one thread for each successor of s,
- in addition, $\mathcal{A}$ verifies that t' conforms to the invariants $I(s)$ for $s \in S$ and the constraints induced by the edge-labeling function L' (e.g. for an edge $(s, s') \in R$ s.t. $(reset, x) \notin L'(s, s')$, the value of x in s is the same as that in s').

Note that during the construction, for the convenience of the presentation, we use the arbitrary positive Boolean combinations of states in the definition of δ. Auxiliary states can be introduced to transform δ into the standard definition. Specifically, $\mathcal{A}$ is constructed as follows.

- Q includes all the states occurring in the definition of δ. The intuitive meaning of the states is the same as that in the proof of Proposition 4.18.
- q_0 is the initial state.
- δ is defined as follows.
  For each state $s \in S$, a formula $\varphi_{L(s)}$ can be defined as in the proof of Proposition 4.18 to describe the satisfaction of (parameterized) atomic propositions.
  - $\delta(q_0) = (s_1, 0) \vee (s_2, 0) \vee \dots \vee (s_r, 0)$.
  - For every $s \in S$, let $R(s) = \{s'_0, \dots, s'_i\}$, then $\delta((s, 0)) = p' \wedge (s, 0, I(s)) \wedge (s, 0, \varphi_{L(s)}) \wedge ([(s, s'_1, 0) \wedge \dots \wedge (s, s'_i, 0)] \vee (s, -, 0))$. Intuitively, when a thread is in the state $(s, 0)$, either for each successor $s'_{i'}$ of s (where $0 \leq i' \leq i$), a thread is created and the state is changed to $((s, s'_{i'}), 0)$, or $(s, -, 0)$ is chosen.
  - For every $s \in S$ and $j \in [K-1]$, $\delta((s, -, j)) = \triangledown_0 (s, -, j+1)$, moreover, $\delta((s, -, K-1)) = \bigwedge\limits_{0 \leq j' < k} \overline{\triangledown_{j'}}?$.
  - For every $s \in S$ and every subformula $\psi = \psi_1 \ op\ \psi_2$ of $I(s)$ (where $op = \vee, \wedge$), $\delta((s, 0, \psi)) = (s, 0, \psi_1)\ op\ (s, 0, \psi_2)$.
  - For every $s \in S$, every subformula $x_i\ op\ x_{i'}$ (where $op \in \{=, \neq\}$ and $i < i'$) of $I(s)$, and every $j : 0 \leq j < i$, $\delta((s, j, x_i\ op\ x_{i'})) = \triangledown_0 (s, j+1, x_i\ op\ x_{i'}) \wedge \bigwedge\limits_{1 \leq j' < k} \overline{\triangledown_{j'}}?$, $\delta((s, i, x_i\ op\ x_{i'})) = store((s, i, (x_{i'})_{op}))$.
  - For every $s \in S$, $i : 0 \leq i \leq K-1$, $j : 0 \leq j < i$, and $op \in \{=, \neq\}$, $\delta((s, j, (x_i)_{op})) = \triangledown_0 (s, j+1, (x_i)_{op}) \wedge \bigwedge\limits_{1 \leq j' < k} \overline{\triangledown_{j'}}?$, $\delta((s, i, (x_i)_=)) = eq$, $\delta((s, i, (x_i)_{\neq})) = \overline{eq}$.
  - For every $s \in S$ and every subformula $\psi = \psi_1 \wedge \psi_2$ of $\varphi_{L(s)}$, $\delta((s, 0, \psi)) = (s, 0, \psi_1) \wedge (s, 0, \psi_2)$,
  - For every $s \in S$ $i : 1 \leq i \leq l$, $\delta((s, 0, p_i)) = p_i$, $\delta((s, 0, \neg p_i)) = \neg p_i$.
  - For every $s \in S$, $i : 1 \leq i \leq m$, $j : 0 \leq j \leq K-1$ and $j' : 0 \leq j' < j$, $\delta((s, j', \tau_i(x_j))) = \triangledown_0 (s, j'+1, \tau_i(x_j)) \wedge \bigwedge\limits_{1 \leq j'' < k} \overline{\triangledown_{j''}}?$, $\delta((s, j', \neg \tau_i(x_j))) = \triangledown_0 (s, j'+1, \neg \tau_i(x_j)) \wedge \bigwedge\limits_{1 \leq j'' < k} \overline{\triangledown_{j''}}?$, moreover, $\delta((s, j, \tau_i(x_j))) = \tau_i$, and $\delta((s, j, \neg \tau_i(x_j))) = \neg \tau_i$.
  - For every $(s, s') \in R$, let $Y = \{x_i \mid (reset, x_i) \notin L'(s, s')\}$, then $\delta(((s, s'), 0)) = \triangledown_0 ((s, s'), 1) \wedge \bigwedge\limits_{1 \leq j < k} \overline{\triangledown_j}? \wedge \bigwedge\limits_{x_i \in Y} ((s, s'), 0, x_i)$. Intuitively, $((s, s'), 0, x_i)$ is used to verify that the value of x_i in the state s' is the same as that of s.
  - For every $s \in S$, let $R(s) = \{s'_0, \dots, s'_i\}$, then for every $i : 0 \leq i' \leq i$ and $j : 0 \leq j < K-1$, $\delta(((s, s'_{i'}), j)) = \triangledown_0 ((s, s'_{i'}), j+1) \wedge \bigwedge\limits_{1 \leq j' < k} \overline{\triangledown_{j'}}?$, $\delta(((s, s'_{i'}), K-1)) = \triangledown_{i'} (s'_{i'}, 0) \wedge \bigwedge\limits_{i < j' < k} \overline{\triangledown_{j'}}?$.
  - For every $(s, s') \in R$, $i : 0 \leq i \leq K-1$ and $j : 0 \leq j < i$, $\delta(((s, s'), j, x_i)) = \triangledown_0 ((s, s'), j+1, x_i) \wedge \bigwedge\limits_{1 \leq j' < k} \overline{\triangledown_{j'}}?$, $\delta(((s, s'), i, x_i)) = store(((s, s'), i, (x_i)_=))$. Intuitively, $((s, s'), i, (x_i)_=)$ means that the data value assigned to x_i on s has been stored in the register, and this value should be equal to that of s'.
  - For every $(s, s') \in R$, $i : 0 \leq i \leq K-1$ and $j : i \leq j < K-1$, $\delta(((s, s'), j, (x_i)_=)) = \triangledown_0 ((s, s'), j+1, (x_i)_=) \wedge \bigwedge\limits_{1 \leq j' < k} \overline{\triangledown_{j'}}?$.
  - For every $s \in S$, let $R(s) = \{s'_0, \dots, s'_i\}$, then for every $i' : 0 \leq i' \leq i$ and $j : 0 \leq j \leq K-1$, $\delta(((s, s'_{i'}), K-1, (x_j)_=)) = \triangledown_{i'} (s'_{i'}, 0, (x_j)_=) \wedge \bigwedge\limits_{i < j' < k} \overline{\triangledown_{j'}}?$. □

Suppose that φ is a normalized VCTL formula. Then $enc(\varphi) = \varphi_1' \wedge \varphi_2'$ with φ_1' and φ_2' defined as follows.

- φ_1' puts restrictions on the occurrences of p' and the atomic propositions from AP,

$$\varphi_1' = p' \wedge G\left[p' \rightarrow \begin{array}{c} \bigwedge\limits_{p \in AP} [(\bigwedge\limits_{0 \le i \le K-1} (AX)^i p) \vee (\bigwedge\limits_{0 \le i \le K-1} (AX)^i \neg p)] \wedge \\ \bigwedge\limits_{1 \le i \le K-1} (AX)^i \neg p' \wedge (A\overline{X})^K p' \end{array} \right].$$

  Intuitively, φ_1' states that p' occurs in the first position, for every occurrence of p' in some position, p' will occur in the K-th position after it if there is such a position, but does not occur in between, moreover, for every $p \in AP$, either p occurs in all the positions between two adjacent occurrences of p', or occurs in none of them.
- φ_2' is obtained from φ by the following procedure.
  1. Replace every eventual occurrence of $AX\phi$ (resp. $A\overline{X}\phi$, $EX\phi$, $E\overline{X}\phi$) by $p' \wedge (AX)^K \phi$ (resp. $p' \wedge (A\overline{X})^K\phi$, $p' \wedge (EX)^K\phi$, $p' \wedge (E\overline{X})^K\phi$).
  2. Replace every persistent occurrence of $AX\phi$ (resp. $A\overline{X}\phi$, $EX\phi$, $E\overline{X}\phi$) by $p' \rightarrow (AX)^K\phi$ (resp. $p' \rightarrow (A\overline{X})^K\phi$, $p' \rightarrow (EX)^K\phi$, $p' \rightarrow (E\overline{X})^K\phi$).
  3. For every $p \in AP$, replace every eventual occurrence of p (resp. $\neg p$) by $p' \wedge \bigvee\limits_{0 \le i \le K-1} (AX)^i p$ (resp. $p' \wedge \bigvee\limits_{0 \le i \le K-1} (AX)^i \neg p$), and every persistent occurrence of p (resp. $\neg p$) by $p' \rightarrow \bigvee\limits_{0 \le i \le K-1} (AX)^i p$ (resp. $p' \rightarrow \bigvee\limits_{0 \le i \le K-1} (AX)^i \neg p$). The formula $\bigvee\limits_{0 \le i \le K-1} (AX)^i p$ (resp. $\bigvee\limits_{0 \le i \le K-1} (AX)^i \neg p$) states that p (resp. $\neg p$) occurs in one of the next K positions. This is sound since for every $p \in AP$, φ_1' requires that either p occurs in all of them or none of them.
  4. For every $\tau \in T$ and $x \in Var$, replace every eventual occurrence of $\tau(x)$ (resp. $\neg\tau(x)$) by $p' \wedge \bigvee\limits_{0 \le i \le K-1} (AX)^i \tau(x)$ (resp. $p' \wedge \bigvee\limits_{0 \le i \le K-1} (AX)^i \neg\tau(x)$), and replace every persistent occurrence of $\tau(x)$ (resp. $\neg\tau(x)$) by $p' \rightarrow \bigvee\limits_{0 \le i \le K-1} (AX)^i \tau(x)$ (resp. $p' \rightarrow \bigvee\limits_{0 \le i \le K-1} (AX)^i \neg\tau(x)$).
  5. For every $i \in [K]$, replace every eventual occurrence of $x@a_i$ (resp. $\neg x@a_i$) with $p' \wedge (AX)^i x@a'$ (resp. $p' \wedge (AX)^i \neg x@a'$), every persistent occurrence of $x@a_i$ (resp. $\neg x@a_i$) with $p' \rightarrow (AX)^i x@a'$ (resp. $p' \rightarrow (AX)^i \neg x@a_i$).

Theorem 5.6 is proved in the same way as Theorem 4.10, by utilizing the following two results.

Proposition 5.8. *For each VCTL formula φ over k-ary $\mathbb{A}$-attributed data trees, it holds $enc(\mathcal{L}(\varphi)) = \mathcal{L}(enc(\varphi))$.*

Proposition 5.9. *For every $\exists^*$-VCTL sentence φ, if φ is satisfiable over an $\mathbb{A}$-attributed data tree where $\mathbb{A}$ is a singleton, then there is a $(2|\varphi|)$-ary $\mathbb{A}$-attributed data tree satisfying φ.*

Proof. We first introduce some notations.

Let φ be an $\exists^*$-VCTL formula, define the closure of φ, denoted by $cl(\varphi)$, as the minimum subset of $\exists^*$-VCTL formulae satisfying the following conditions.

- $\varphi \in cl(\varphi)$,
- for every $p \in AP$ occurring in φ, it holds $p, \neg p \in cl(\varphi)$,
- for every $\tau(x) \in T \times var(\varphi)$ occurring in φ, it holds $\tau(x), \neg\tau(x) \in cl(\varphi)$,
- for every $\psi \in cl(\varphi)$ s.t. $\psi := \psi_1 \wedge \psi_2$ or $\varphi := \psi_1 \vee \psi_2$, it holds $\psi_1, \psi_2 \in cl(\varphi)$,
- for every $\psi \in cl(\varphi)$ s.t. $\psi := EX\psi_1$ or $\psi := AX\psi_1$ or $\psi := E\overline{X}\psi_1$ or $\psi := A\overline{X}\psi_1$, it holds $\psi_1 \in cl(\varphi)$,
- for every $\psi \in cl(\varphi)$ s.t. $\psi := E(\psi_1 U \psi_2)$ or $\psi := E(\psi_1 R \psi_2)$, it holds $\psi_1, \psi_2, EX\psi \in cl(\varphi)$,
- for every $\psi \in cl(\varphi)$ s.t. $\psi := A(\psi_1 U \psi_2)$ or $\psi := A(\psi_1 R \psi_2)$, it holds $\psi_1, \psi_2, AX\psi \in cl(\varphi)$,
- for every $\psi \in cl(\varphi)$ s.t. $\psi := \exists x.\psi_1$, it holds $\psi_1 \in cl(\varphi)$.

From the above definition, we know that $|cl(\varphi)| \le 2|\varphi|$.

Let φ be an $\exists^*$-VCTL sentence and $t = (Z, L)$ be a data tree s.t. $t \models \varphi$. Without loss of generality, we assume that for every variable $x \in var(\varphi)$, x is quantified only once.

In the following, we will construct a $(2|\varphi|)$-ary data tree t' from t and φ s.t. $t' \models \varphi$. Intuitively, t' is constructed from t in a top-down manner by

1. labeling each node z of t with the pairs $(\psi, \lambda) \in cl(\varphi) \times (Var \rightarrow \mathbb{D})$ s.t. $\lambda : free(\psi) \rightarrow \mathbb{D}$ and $t|_z \models_\lambda \psi$,
2. for each node t and each $\psi \in cl(\varphi)$, selecting exactly one assignment function λ s.t. (ψ, λ) belongs to the label of t (this is sufficient to make φ satisfied, due to the fact that only existential quantifications are used in φ),

3. as a result of this special property, selecting a bounded number of subtrees for each node of t and removing all the else from t.

In the following, we present a detailed construction of t' from t. Before that, we first introduce some definitions and notations.

Define a new labeling function L' for nodes in Z as follows: for every $z \in Z$, $L'(z)$ is the set of pairs $(\psi, \lambda) \in cl(\varphi) \times (Var \to \mathbb{D})$ s.t. $\lambda : free(\psi) \to \mathbb{D}$ and $t|_z \models_\lambda \psi$.

Let $z \in Z$ and $\Phi \subseteq L'(z)$. Then Φ is said to be *functional* if the following two conditions hold,

- for every $\psi \in cl(\varphi)$, there is at most one λ s.t. $(\psi, \lambda) \in \Phi$,
- for every formula $(\psi_1, \lambda_1), (\psi_2, \lambda_2) \in \Phi$, $\lambda_1|_{free(\psi_1)\cap free(\psi_2)} = \lambda_2|_{free(\psi_1)\cap free(\psi_2)}$.

Suppose $z \in Z$ and $\Phi \subseteq L'(z)$ s.t. Φ is nonempty and functional. Then $\Phi' \subseteq L'(z)$ is said to be a *completion* of Φ with respect to z, denoted by $\Phi \to_{z,comp} \Phi'$, if Φ' can be constructed from Φ by the following procedure.

1. Initially, let $\Phi' = \Phi$.
2. Repeat the following procedure until all the formulae in Φ' are one of the following forms: $p, \neg p, \tau(x), \neg\tau(x)$, $EX\psi$, $AX\psi$, $E\overline{X}\psi$, $A\overline{X}\psi$.
   For every $(\psi, \lambda) \in \Phi'$, let $\Phi' = \Phi' \setminus \{(\psi, \lambda)\}$, and do one of the following.
   - $\psi = \psi_1 \vee \psi_2$:
     – if $(\psi_1, \lambda|_{free(\psi_1)}) \in L'(z)$, then $\Phi' = \Phi' \cup \{(\psi_1, \lambda|_{free(\psi_1)})\}$ (if there is already $(\psi_1, \lambda') \in \Phi'$ before adding $\{(\psi_1, \lambda|_{free(\psi_1)})\}$ into Φ', then from the functionality of Φ and the construction, it must be the case that $\lambda' = \lambda|_{free(\psi_1)}$, therefore, the functionality of Φ' is preserved, the same remark applies below),
     – otherwise, $\Phi' = \Phi' \cup \{(\psi_2, \lambda|_{free(\psi_2)})\}$,
   - $\psi = \psi_1 \wedge \psi_2$: $\Phi' = \Phi' \cup \{(\psi_1, \lambda|_{free(\psi_1)}), (\psi_2, \lambda|_{free(\psi_2)})\}$,
   - $\psi = E(\psi_1 U \psi_2)$:
     – if $(\psi_2, \lambda|_{free(\psi_2)}) \in L'(z)$, then $\Phi' = \Phi' \cup \{(\psi_2, \lambda|_{free(\psi_2)})\}$,
     – otherwise, $\Phi' = \Phi' \cup \{(\psi_1, \lambda|_{free(\psi_1)}), (EX\psi, \lambda)\}$,
   - $\psi = A(\psi_1 U \psi_2)$:
     – if $(\psi_2, \lambda|_{free(\psi_2)}) \in L'(z)$, then $\Phi' = \Phi' \cup \{(\psi_2, \lambda|_{free(\psi_2)})\}$,
     – otherwise, $\Phi' = \Phi' \cup \{(\psi_1, \lambda|_{free(\psi_1)}), (AX\psi, \lambda)\}$,
   - $\psi = E(\psi_1 R \psi_2)$:
     – if $(\psi_1, \lambda|_{free(\psi_1)}) \in L'(z)$, then $\Phi' = \Phi' \cup \{(\psi_1, \lambda|_{free(\psi_1)}), (\psi_2, \lambda|_{free(\psi_2)})\}$,
     – otherwise, $\Phi' = \Phi' \cup \{(\psi_2, \lambda|_{free(\psi_2)}), (EX\psi, \lambda)\}$,
   - $\psi = A(\psi_1 R \psi_2)$:
     – if $(\psi_1, \lambda|_{free(\psi_1)}) \in L'(z)$, then $\Phi' = \Phi' \cup \{(\psi_1, \lambda|_{free(\psi_1)}), (\psi_2, \lambda|_{free(\psi_2)})\}$,
     – otherwise, $\Phi' = \Phi' \cup \{(\psi_2, \lambda|_{free(\psi_2)}), (AX\psi, \lambda)\}$,
   - $\psi = \exists x.\psi_1$: If there does not exist $d \in \mathbb{D}$ s.t. $(\psi_1, \lambda[d/x]) \in \Phi'$, select a data value $d \in \mathbb{D}$ s.t. $(\psi_1, \lambda[d/x]) \in L'(z)$, let $\Phi' = \Phi' \cup \{(\psi_1, \lambda[d/x])\}$.

Since in the above construction, for every $\exists x.\psi_1$, only one instantiation of x is allowed, it follows that if Φ is functional and $\Phi \to_{z,comp} \Phi'$, then Φ' is functional as well.

Now we are ready to construct $t' = (Z', L'')$ from t. The construction is done in a top-down manner. During the construction, two functions F and F' are also constructed.

1. Select an assignment function λ s.t. $(\varphi, \lambda) \in L'(z)$. Let $\varepsilon \in Z'$, $L''(\varepsilon) = L(\varepsilon)$, and $F(\varepsilon) = \{(\varphi, \lambda)\}$.
2. Repeat the following procedure.
   For every $z \in Z'$ s.t. z is currently a leaf in Z', let $F'(z)$ be a completion of $F(z)$ with respect to z. Do one of the following.
   (a) If $F'(z)$ contains neither elements of the form $(AX\psi_1, \lambda)$ nor elements of the form $(EX\psi_1, \lambda)$, then z is a leaf in t'.
   (b) If $F'(z)$ contains elements of the form $(AX\psi_1, \lambda)$ (this implies that z cannot be a leaf in t'), but neither elements of the form $(EX\psi_1, \lambda)$ nor elements of the form $(E\overline{X}\psi_1, \lambda)$, then add $z0$ into Z', and $L''(z0) = L(z0)$.
   (c) If $F'(z)$ contains elements of the form $(AX\psi_1, \lambda)$ or $(EX\psi_1, \lambda)$ (this implies that z cannot be a leaf in t'), moreover, $F'(z)$ contains elements of the form $(EX\psi_1, \lambda)$ or $(E\overline{X}\psi_1, \lambda)$, then for every $(EX\psi_1, \lambda) \in F'(z)$ or $(E\overline{X}\psi_1, \lambda) \in F'(z)$, select a child of z in t, say zi, s.t. $(\psi_1, \lambda) \in L'(zi)$. We can assume that all these selected children are distinct from each other, since otherwise we can just copy the subtrees of z to satisfy this property. Let $idx_{EX\psi_1}$ (resp. $idx_{E\overline{X}\psi_1}$) denote the natural number s.t. $z\ idx_{EX\psi_1}$ (resp. $z\ idx_{E\overline{X}\psi_1}$) is the child of z selected for $EX\psi_1$ (resp. $E\overline{X}\psi_1$) above. Suppose that there are r elements in $F'(z)$ of the form $(EX\psi_1, \lambda)$ or $(E\overline{X}\psi_1, \lambda)$. Without loss of generality, we assume that the set of indices $idx_{EX\varphi_1}$ and $idx_{E\overline{X}\varphi_1}$ is $[r]$. Then add $z0, \ldots, z(r-1)$ into Z', let $L''(zi) = L(zi)$ for every $i : 0 \le i \le r-1$, and set $F(z\ idx_{EX\psi_1})$ (resp. $F(z\ idx_{E\overline{X}\psi_1})$) as $\{(\psi_1, \lambda)\} \cup \{(\psi', \lambda') \mid (AX\psi', \lambda') \in F'(z)$ or $(A\overline{X}\psi', \lambda') \in F'(z)\}$. Note that all these sets $F(z\ idx_{EX\psi_1})$ and $F(z\ idx_{E\overline{X}\psi_1})$ are functional, since $F'(z)$ is.

The above procedure terminates, since $t \models \varphi$ and all the leaves z of t satisfy that $L'(z)$ contains neither elements of the form $(AX\psi_1, \lambda)$ nor elements of the form $(EX\psi_1, \lambda)$.

From the construction, we know that the arities of nodes in t' are bounded by $2|\varphi|$ and $t' \models \varphi$. □

Proof of Theorem 5.6. Let φ be a NN-$\exists^*$-VCTL sentence. From the definition of $enc(\varphi)$, it is not hard to observe that $enc(\varphi)$ is also a NN-$\exists^*$-VCTL sentence. In addition, from Proposition 5.8, we know that $enc(\mathcal{L}(\varphi)) = \mathcal{L}(enc(\varphi))$. Therefore, it is sufficient to consider the satisfiability of $enc(\varphi)$ over $\mathbb{A}'$-attributed data trees where $\mathbb{A}'$ is a singleton. Moreover, from Proposition 5.9, it is sufficient to consider the satisfiability of $enc(\varphi)$ over $(2|\varphi|)$-ary $\mathbb{A}'$-attributed data trees.

Similar to the proof of Theorem 4.10, we can prove by an induction on the structure of formulae that from $enc(\varphi)$, an equivalent ATRA $\mathcal{A}_{enc(\varphi)}$ over $(2|\varphi|)$-ary $\mathbb{A}'$-attributed data trees can be constructed. The decidability then follows from Theorem 2.8.

The lower bound proof is also similar to that of NN-$\exists^*$-VLTL. □

Theorem 5.10. *The model checking problem of NN-$\forall^*$-VCTL is decidable and non-primitive recursive.*

Proof. Let $\mathcal{K}$ be a VKS and φ be a NN-$\forall^*$-VCTL sentence. Then $\mathcal{K} \not\models \varphi$ iff there is a computation tree t of $\mathcal{K}$ s.t. $t \models \overline{\varphi}$ iff $\mathcal{T}(\mathcal{K}) \cap \mathcal{L}(\overline{\varphi}) \neq \emptyset$ iff $enc(\mathcal{T}(\mathcal{K})) \cap enc(\mathcal{L}(\overline{\varphi})) \neq \emptyset$ iff $enc(\mathcal{T}(\mathcal{K})) \cap \mathcal{L}(enc(\overline{\varphi})) \neq \emptyset$.

From Proposition 5.7, we know that there is an ATRA $\mathcal{A}$ s.t. $\mathcal{L}(\mathcal{A}) = enc(\mathcal{T}(\mathcal{K}))$.

Let k be the maximum number of successors of states in $\mathcal{K}$. Since $enc(\overline{\varphi})$ is a NN-$\exists^*$-VCTL formula, we can prove by an induction on the structure of formulae that from $enc(\overline{\varphi})$, an equivalent ATRA $\mathcal{A}_{enc(\overline{\varphi})}$ over k-ary $\mathbb{A}'$-attributed data trees can be constructed (where $\mathbb{A}'$ is a singleton).

Then $\mathcal{K} \not\models \varphi$ iff $\mathcal{L}(\mathcal{A}) \ \cap \ \mathcal{L}(\mathcal{A}_{enc(\overline{\varphi})}) \neq \emptyset$.

Since ATRAs are closed under intersection, the decidability then follows from Theorem 2.8.

The non-primitive lower bound follows from the fact that the satisfiability of NN-$\exists^*$-VLTL can be reduced in polynomial time to the model checking problem of NN-$\forall^*$-VCTL, by using the idea in the proof of Theorem 5.5. □

5.2.2. Existential path quantifiers for VCTL

Theorem 5.11. *The satisfiability problem of EVCTL is in NEXPTIME.*

For the proof of Theorem 5.11, we first state and prove several facts about EVCTL.

From the fact that EVCTL formulae contain only existential path quantifiers, we have the following observation.

Lemma 5.12. *Let t be an $\mathbb{A}$-attributed data tree, φ be an EVCTL formula, and $\lambda : free(\varphi) \to \mathbb{D}$ s.t. $t \models_\lambda \varphi$. Then the following two facts hold.*

1. *Let t' be an $\mathbb{A}$-attributed data tree s.t. there is a leaf-preserving embedding of t into t'. Then $t' \models_\lambda \varphi$.*
2. *Let η be an* injective *partial function from $\mathbb{D}$ to $\mathbb{D}$ s.t. the domain of η includes all the data values occurring in (t, λ), moreover, for each data value d occurring in λ, $\eta(d) = d$. Then $\eta(t) \models_\lambda \varphi$, where $\eta(t)$ is obtained from t by replacing each data value d with $\eta(d)$.*

Intuitively, the argument for the first fact in Lemma 5.12 is as follows: Since there is a leaf-preserving embedding η of t into t', for each path π of t, there is a corresponding path $\eta(\pi)$ in t'. Since φ contains only existential path quantifiers, and $t \models_\lambda \varphi$, we can restrict our attention to the paths $\eta(\pi)$ in t' (where π is a path in t) and ensure $t' \models_\lambda \varphi$.

Then we show that for the satisfaction of EVCTL formulae, a bounded number of data values are sufficient.

Lemma 5.13. *Let φ be an EVCTL formula, t be an $\mathbb{A}$-attributed data tree, and $\lambda : free(\varphi) \to \mathbb{D}$ s.t. $t \models_\lambda \varphi$. Then an $\mathbb{A}$-attributed data tree t' can be constructed from (t, λ) s.t. $t' \models_\lambda \varphi$, the label of the root of t' is the same as that of t, all the data values in t' also occur in t, and (t', λ) contains at most $(|\mathbb{A}| + 1)|\varphi|$ data values.*

Lemma 5.13 cannot be extended to data ω-trees. For instance, let $\varphi = \forall x.\ EFp(x)$. Then φ is ω-satisfiable. Nevertheless, any data ω-tree satisfying φ has to contain all the data values from $\mathbb{D}$.

Proof of Lemma 5.13. We construct t' from t by an induction on the syntax of EVCTL formulae. Recall that EVCTL formulae are defined by the following rules,

$$\begin{array}{rl} \varphi := & p \mid \neg p \mid \tau(x) \mid \neg\tau(x) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid EX\varphi \mid E\overline{X}\varphi \\ & \mid E(\varphi U \varphi) \mid E(\varphi R \varphi) \mid \exists x.\varphi \mid \forall x.\varphi \end{array},$$

where $p \in AP$, $\tau \in T$ and $x \in Var$.

The basic idea of the construction is as follows. For each formula φ, suppose the desired subtrees have been constructed for the subformulae of φ, then we glue these subtrees in a proper way to obtain a tree which still satisfies φ (attributed to Lemma 5.12). Moreover, when dealing with the universal quantifications, some data values in these subtrees are renamed in order to bound the number of data values occurring in t'.

The induction base $\varphi := p, \neg p, \tau(x), \neg\tau(x), x@a, \neg x@a$: Trivial.

The induction step.

$\varphi := \varphi_1 \vee \varphi_2$: Suppose $t \models_\lambda \varphi_1 \vee \varphi_2$, then $t \models_\lambda \varphi_1$ or $t \models_\lambda \varphi_2$. If $t \models_\lambda \varphi_1$, then by the induction hypothesis, an $\mathbb{A}$-attributed data tree t_1 can be constructed from $(t, \lambda|_{free(\varphi_1)})$ s.t. $t_1 \models_{\lambda|_{free(\varphi_1)}} \varphi_1$, the root label of t_1 is the same as that of t, all the data values of t_1 also occur in t, and $(t_1, \lambda|_{free(\varphi_1)})$ contains at most $(|\mathbb{A}|+1)|\varphi_1|$ data values. Let $t' = t_1$. Then $t' \models_\lambda \varphi$, the root label of t' is the same as that of t, all the data values in t' also occur in t, and (t', λ) contains at most $(|\mathbb{A}|+1)|\varphi_1| + |free(\varphi_2) \setminus free(\varphi_1)| \leq (|\mathbb{A}|+1)(|\varphi_1|+|\varphi_2|) \leq (|\mathbb{A}|+1)|\varphi|$ data values. The situation that $t \models_\lambda \varphi_2$ can be discussed similarly.

$\varphi := \varphi_1 \wedge \varphi_2$: Suppose $t \models_\lambda \varphi_1 \wedge \varphi_2$. Then $t \models_\lambda \varphi_1$ and $t \models_\lambda \varphi_2$. By the induction hypothesis, t_1 and t_2 can be constructed from $(t, \lambda|_{free(\varphi_1)})$ and $(t, \lambda|_{free(\varphi_2)})$ s.t. for each $i = 1, 2$, $t_i \models_{\lambda|_{free(\varphi_i)}} \varphi_i$, the root label of t_i is the same as that of t, all the data values in t_i also occur in t, and $(t_i, \lambda|_{free(\varphi_i)})$ contains at most $(|\mathbb{A}|+1)|\varphi_i|$ data values. Let t' be the data tree obtained from t_1 by adding all the subtrees of the root of t_2 as the new subtrees of the root of t_1 (with the original subtrees of the root of t_1 untouched). From the fact that t_1 and t_2 are substructures of t' and Lemma 5.12, we deduce that $t' \models_{\lambda|_{free(\varphi_1)}} \varphi_1$ and $t' \models_{\lambda|_{free(\varphi_2)}} \varphi_2$. Therefore, $t' \models_\lambda \varphi_1 \wedge \varphi_2$. In addition, the root label of t' is the same as that of t, all the data values in t' also occur in t, and (t', λ) contains at most $(|\mathbb{A}|+1)(|\varphi_1|+|\varphi_2|) \leq (|\mathbb{A}|+1)|\varphi|$ data values.

$\varphi := EX\varphi_1$: Let $t \models_\lambda EX\varphi_1$. Then there is a child of the root of t, say the node i, s.t. $t|_i \models_\lambda \varphi_1$. By the induction hypothesis, an $\mathbb{A}$-attributed data tree t' can be constructed from $(t|_i, \lambda)$ s.t. $t' \models_\lambda \varphi_1$, the root label of t' is the same as that of $t|_i$, all the data values in t' occur in $t|_i$, and (t', λ) contains at most $(|\mathbb{A}|+1)|\varphi_1|$ data values. Let t'' be the data tree obtained from t' by adding the root of t as the parent of the root of t'. Then $t' \models_\lambda EX\varphi_1$, the root of t' is the same as that of t, all the data values in t' also occur in t, and (t', λ) contains at most $(|\mathbb{A}|+1)|\varphi_1| + |\mathbb{A}| \leq (|\mathbb{A}|+1)|\varphi|$ data values.

$\varphi := E\overline{X}\varphi_1$: Let $t \models_\lambda E\overline{X}\varphi_1$. Then either t is a just a single node or otherwise $t \models_\lambda EX\varphi_1$. The first case is trivial. The second case can be discussed similarly to $\varphi := EX\varphi_1$.

$\varphi := E(\varphi_1 U \varphi_2)$: Suppose $t \models_\lambda \varphi$. Then there are a path $\pi = \pi_0 \dots \pi_n$ in t and $i : 0 \leq i \leq n$ satisfying that $t|_{\pi_i} \models_\lambda \varphi_2$ and for every $j : 0 \leq j < i$, $t|_{\pi_j} \models_\lambda \varphi_1$. We distinguish between $i = 0$ and $i > 0$.

- If $i = 0$, then $t \models_\lambda \varphi_2$. By the induction hypothesis, an $\mathbb{A}$-attributed data tree t' can be constructed from $(t, \lambda|_{free(\varphi_2)})$ s.t. $t' \models_{\lambda|_{free(\varphi_2)}} \varphi_2$, the root label of t' is the same as that of t, all the data values in t' also occur in t, and $(t', \lambda|_{free(\varphi_2)})$ contains at most $(|\mathbb{A}|+1)|\varphi_2|$ data values. Then (t', λ) is the desired pair for φ.
- If $i > 0$, then $t|_{\pi_i} \models_\lambda \varphi_2$ and $t|_{\pi_0} \models_\lambda \varphi_1$. By the induction hypothesis, the data trees t_1 and t_2 can be constructed from $(t|_{\pi_0}, \lambda|_{free(\varphi_1)})$ and $(t|_{\pi_i}, \lambda|_{free(\varphi_2)})$ respectively s.t. $t_1 \models_{\lambda|_{free(\varphi_1)}} \varphi_1$, $t_2 \models_{\lambda|_{free(\varphi_2)}} \varphi_2$, the root labels of t_1 and t_2 are the same as that of $t|_{\pi_0}$ and $t|_{\pi_i}$ respectively, all the data values in t_1 and t_2 also occur in $t|_{\pi_0}$ and $t|_{\pi_i}$ respectively, in addition, $(t|_{\pi_0}, \lambda|_{free(\varphi_1)})$ and $(t|_{\pi_i}, \lambda|_{free(\varphi_2)})$ contain respectively at most $(|\mathbb{A}|+1)|\varphi_1|$ and $(|\mathbb{A}|+1)|\varphi_2|$ data values. Let t' be the data tree obtained from t, t_1 and t_2 as follows: t' contains a path of length (number of nodes) $i+1$, say the sequence $(\varepsilon, 0, 0^2, \dots, 0^i)$, s.t. a copy of t_2 is attached to 0^i, and for every $j : 0 \leq j < i$, a copy of t_1 is attached to 0^j. Note that the root label of t' is the same as that of t_1, thus the same as that of $t|_{\pi_0} = t$. From Lemma 5.12 and the fact that t_2 is a substructure of $t'|_{0^i}$ and t_1 is a substructure of $t'|_{0^j}$ for every $j : 0 \leq j < i$, we know that $t'|_{0^i} \models_{\lambda|_{free(\varphi_2)}} \varphi_2$, and $t'|_{0^j} \models_{\lambda|_{free(\varphi_1)}} \varphi_1$ for every $j : 0 \leq j < i$. Therefore, $t' \models_\lambda E(\varphi_1 U \varphi_2)$, the root label of t' is the same as that of t, all the data values in t' also occur in t, and (t', λ) contains at most $(|\mathbb{A}|+1)(|\varphi_1|+|\varphi_2|) \leq (|\mathbb{A}|+1)|\varphi|$ data values.

$\varphi := E(\varphi_1 R \varphi_2)$: Suppose $t \models_\lambda \varphi$. Then there is a path $\pi = \pi_0 \dots \pi_n$ in t s.t. either $t|_{\pi_i} \models_\lambda \varphi_2$ for every $i : 0 \leq i \leq n$, or there is $i : 0 \leq i \leq n$ s.t. $t|_{\pi_i} \models_\lambda \varphi_1$ and for every $j : 0 \leq j \leq i$, $t|_{\pi_j} \models_\lambda \varphi_2$.

- **The case** $t|_{\pi_i} \models_\lambda \varphi_2$ **for every** $i : 0 \leq i \leq n$. If $n = 0$, that is, t is a single node, then the argument is trivial. Otherwise, by the induction hypothesis, a data tree t'_1 can be constructed from $(t, \lambda|_{free(\varphi_2)})$ s.t. $t'_1 \models_{\lambda|_{free(\varphi_2)}} \varphi_2$, the root label of t'_1 is the same as that of t, all the data values in t'_1 also occur in t, and $(t'_1, \lambda|_{free(\varphi_2)})$ contains at most $(|\mathbb{A}|+1)|\varphi_2|$ data values. Let t' be the data tree obtained from t and t'_1 as follows: t' contains a path $(\varepsilon, 0)$ s.t. a copy of t'_1 is attached to ε, and the label of 0 is the same label as that of π_n in t (thus 0 is a leaf in t'). Then from $t|_{\pi_n} \models_\lambda \varphi_2$, we know that $t'|_0 \models_\lambda \varphi_2$. Moreover, from Lemma 5.12 and the fact that t'_1 is a substructure of t', we deduce that $t' \models_\lambda \varphi_2$. Therefore, $t' \models_\lambda E(\varphi_1 R \varphi_2)$, the root label of t' is the same as that of t, all the data values in t' also occur in t, and t' contains at most $(|\mathbb{A}|+1)|\varphi_2| + |\mathbb{A}| \leq (|\mathbb{A}|+1)|\varphi|$ data values.
- **The case that there is** $i : 0 \leq i \leq n$ **s.t.** $t|_{\pi_i} \models_\lambda \varphi_1$ **and for every** $j : 0 \leq j \leq i$, $t|_{\pi_j} \models_\lambda \varphi_2$. We distinguish between the following two situations.
  - If $i = 0$, then $t \models_\lambda \varphi_1 \wedge \varphi_2$. Then the argument is similar to the case $\varphi = \varphi_1 \wedge \varphi_2$ above.
  - If $i > 0$, then by the induction hypothesis, a data tree t'_2 can be constructed from $(t, \lambda|_{free(\varphi_2)})$ s.t. $t'_2 \models_{\lambda|_{free(\varphi_2)}} \varphi_2$, the root label of t'_2 is the same as that of t, all the data values in t'_2 also occur in t, and t'_2 contains at most $(|\mathbb{A}|+1)|\varphi_2|$ data values. Similarly, a data tree t''_2 can be constructed from $(t|_{\pi_i}, \lambda|_{free(\varphi_2)})$ s.t. $t''_2 \models_{\lambda|_{free(\varphi_2)}} \varphi_2$, the root label of t''_2

is the same as $t|_{\pi_i}$, all the data values in t_2'' also occur in $t|_{\pi_i}$, and $(t_2'', \lambda|_{free(\varphi_2)})$ contain at most $(|\mathbb{A}|+1)|\varphi_2|$ data values. Moreover, a data tree t_1' can be constructed from $(t|_{\pi_i}, \lambda|_{free(\varphi_1)})$ s.t. $t_1' \models_{\lambda|_{free(\varphi_1)}} \varphi_1$, the root label of t_1' is the same as $t|_{\pi_i}$, all the data values in t_1' also occur in $t|_{\pi_i}$, and $(t_1', \lambda|_{free(\varphi_1)})$ contain at most $(|\mathbb{A}|+1)|\varphi_1|$ data values. Let D be a set of $(|\mathbb{A}|+1)|\varphi_2| + |\mathbb{A}|$ data values that includes all the data values occurring in $(t_2', \lambda|_{free(\varphi_2)})$ and those occurring in the root of t_2''. Let η be an injective partial function from $\mathbb{D}$ to $\mathbb{D}$ s.t.

* the domain of η is $D((t_2'', \lambda|_{free(\varphi_2)}))$ (i.e., the set of data values occurring in $(t_2'', \lambda|_{free(\varphi_2)})$),
* η is the identity function when restricted to the range of $\lambda|_{free(\varphi_2)}$, and the set of data values occurring in the root of t_2'',
* the range of η is a subset of D.

Such a function exists due to the fact that $((t_2'', \lambda|_{free(\varphi_2)}))$ contains at most $(|\mathbb{A}|+1)|\varphi_2|$ data values. From Lemma 5.12, we deduce that $\eta(t_2'') \models_{\lambda|_{free(\varphi_2)}} \varphi_2$. Let t' be the data tree obtained from $t, \eta(t_2''), t_1'$ as follows: t' contains a path $(\varepsilon, 0)$ s.t. a copy of t_2' is attached to ε, and a copy of t_1' as well as a copy of $\eta(t_2'')$ are attached to 0 (note that the root label of $\eta(t_2'')$ is the same as that of t_1'). From Lemma 5.12 again, we know that $t'|_0 \models_\lambda \varphi_1 \wedge \varphi_2$ and $t' \models_{\lambda|_{free(\varphi_1)}} \varphi_1$. Therefore, we conclude that $t' \models_\lambda E(\varphi_1 R \varphi_2)$, the root label of t' is the same as that of t, all the data values in t' also occur in t, and (t', λ) contains at most $|D| + (|\mathbb{A}|+1)|\varphi_1| = (|\mathbb{A}|+1)|\varphi_2| + |\mathbb{A}| + (|\mathbb{A}|+1)|\varphi_1| \leq (|\mathbb{A}|+1)|\varphi|$ data values.

$\varphi := \exists x.\ \varphi_1$: Suppose $t \models_\lambda \exists x.\ \varphi_1$. Then there is $d \in \mathbb{D}$ s.t. $t \models_{\lambda[d/x]} \varphi_1$. By the induction hypothesis, a data tree t_1 can be constructed from $(t, \lambda[d/x])$ s.t. $t_1 \models_{\lambda[d/x]} \varphi_1$, the root label of t_1 is the same as that of t, all the data values in t_1 also occur in t, and $(t_1, \lambda[d/x])$ contains at most $(|\mathbb{A}|+1)|\varphi_1|$ data values. Then (t_1, λ) is the desired pair for φ.

$\varphi := \forall x.\ \varphi_1$: Suppose $t \models_\lambda \forall x.\ \varphi_1$. Then for every $d \in \mathbb{D}$, $t \models_{\lambda[d/x]} \varphi_1$. If (t, λ) contains less than $(|\mathbb{A}|+1)|\varphi_1|$ data values, then we are done. Otherwise, let D be a subset of $(|\mathbb{A}|+1)|\varphi_1|$ data values occurring in (t, λ) s.t. D contains all the data values occurring in the root of t and in λ. Suppose $D = \{d_1, \ldots, d_{(|\mathbb{A}|+1)|\varphi_1|}\}$. In addition, let $d_0 \in \mathbb{D}$ s.t. $d_0 \notin D$ and d_0 does not occur in (t, λ). Then for each $i : 0 \leq i \leq (|\mathbb{A}|+1)|\varphi_1|$, $t \models_{\lambda[d_i/x]} \varphi_1$. From the induction hypothesis, for each $i : 0 \leq i \leq (|\mathbb{A}|+1)|\varphi_1|$, a data tree t_i' can be constructed from $(t, \lambda[d_i/x])$ s.t. $t_i' \models_{\lambda[d_i/x]} \varphi_1$, the root label of t_i' is the same as that of t, all the data values in t_i' also occur in t, and $(t_i', \lambda[d_i/x])$ contains at most $(|\mathbb{A}|+1)|\varphi_1|$ data values. Since $|D| = (|\mathbb{A}|+1)|\varphi_1|$, for each $i : 0 \leq i \leq (|\mathbb{A}|+1)|\varphi_1|$, there is an injective partial function η_i s.t.

1. the domain of η_i includes all the data values occurring in $(t_i', \lambda[d_i/x])$,
2. η_i is the identity function when restricted to the set of data values occurring in $\lambda[d_i/x]$ as well as in the root of t_i' (note that the root label of t_i' is the same as that of t),
3. the range of η_i is a subset of $D \cup \{d_0\}$,
4. the set of data values occurring in $\eta_i(t_i')$ is a subset of D (thus d_0 does not occur in $\eta_i(t_i')$).

The function η_i's exist. For instance, since $(t_0', \lambda[d_0/x])$ contains at most $(|\mathbb{A}|+1)|\varphi_1|$ data values, it follows that except d_0 and the data values occurring in the root of t_0', there are at most $(|\mathbb{A}|+1)(|\varphi_1|-1)$ additional data values from $(t_0', \lambda[d_0/x])$; therefore, it is possible to define an injective partial function η_0 to map those data values into D.

Then from Lemma 5.12, $\eta_i(t_i') \models_{\lambda[d_i/x]} \varphi_1$. Let t'' be the data tree obtained from $\eta_i(t_0'), \ldots, \eta_i(t'_{(|\mathbb{A}|+1)|\varphi_1|})$ by merging their roots (recall that their root labels are the same), that is, all the subtrees of the roots of $\eta_i(t_0'), \ldots, \eta_i(t'_{(|\mathbb{A}|+1)|\varphi_1|})$ are the subtrees of the root in t''. We claim that $t'' \models_\lambda \forall x.\ \varphi_1$. At first, for every d_i with $i : 0 \leq i \leq (|\mathbb{A}|+1)|\varphi_1|$, from $\eta_i(t_i') \models_{\lambda[d_i/x]} \varphi_1$ and Lemma 5.12, we deduce that $t'' \models_{\lambda[d_i/x]} \varphi_1$. Let $d \notin \{d_0, \ldots, d_{(|\mathbb{A}|+1)|\varphi_1|}\}$. Since $t_0' \models_{\lambda[d_0/x]} \varphi_1$ and neither d nor d_0 occurs in $\eta_0(t_0')$, assigning d to x has the same impact as assigning d_0 to x for the satisfaction of φ_1 on $\eta_0(t_0')$. Therefore, $\eta_0(t_0') \models_{\lambda[d/x]} \varphi_1$. From Lemma 5.12 again, we deduce that $t'' \models_{\lambda[d/x]} \varphi_1$. From the fact that d is an arbitrary data value not in $\{d_0, \ldots, d_{(|\mathbb{A}|+1)|\varphi_1|}\}$, we conclude that $t'' \models_\lambda \forall x.\ \varphi_1$, the root label of t'' is the same as that of t, all the data values in t'' also occur in t (recall that the set of data values occurring in $\eta_i(t_i')$ is a subset of D and D is a subset of data values occurring in t), and (t'', λ) contains at most $(|\mathbb{A}|+1)|\varphi_1| + 1 \leq (|\mathbb{A}|+1)|\varphi|$ data values. □

Proof of Theorem 5.11. Suppose φ is an EVCTL sentence. Without loss of generality, we assume that each variable occurring in φ is only quantified once. From Lemma 5.13, we know that if φ is satisfiable, then it is satisfiable over a data tree containing at most $(|\mathbb{A}|+1)|\varphi|$ data values. Let $V = \{d_1, \ldots, d_{(|\mathbb{A}|+1)|\varphi|}\}$.

Construct ECTL formula φ' over $AP \cup (T \times V)$ from φ as follows: Let $\{x_1, \ldots, x_l\}$ be the set of variables occurring in φ. Then φ' is obtained from φ by replacing each $\exists x_i$ (resp. $\forall x_i$) with $\bigvee\limits_{d_i \in V}$ (resp. $\bigwedge\limits_{d_i \in V}$), and every occurrence of $\tau(x_i)$ with (τ, d_i). The size of φ' is exponential over the size of φ.

Since the satisfiability problem of ECTL over labeled trees is decidable in NP [57], it follows that the satisfiability problem of EVCTL is in NEXPTIME. □

Remark 5.14. It is open whether the ω-satisfiability problem of EVCTL is decidable.

5.2.3. Data variable quantifiers in the beginning

Theorem 5.15. *The satisfiability and ω-satisfiability problems of $\exists^*$-VCTL$_{pnf}$ are EXPTIME complete.*

Proof. The upper bound. For every sentence of the form $\exists x_1 ... \exists x_n.\psi$ s.t. ψ is a quantifier free VCTL formula, we only need to consider at most $n+1$ values. It is sufficient to show that there is a data tree (resp. ω-tree) $t = (Z, L)$ satisfying $\exists x_1 ... \exists x_n.\psi$ iff there is a data tree $t' = (Z, L')$ using only $n+1$ different data values s.t. $t' \models \exists x_1 ... \exists x_n.\psi$. Suppose the data tree (resp. ω-tree) t satisfies $\exists x_1 ... \exists x_n.\psi$ and uses values more than $n+1$. W.l.o.g., suppose $x_1, ..., x_n$ take the values from the set $D = \{d_1, ..., d_n\}$ in t, then each $d \in \mathbb{D} \setminus D$ used in t can be replaced by the *same* data value $d' \in \mathbb{D} \setminus D$ without affecting the satisfiability of the formula $\exists x_1 ... \exists x_n.\ \psi$. Thus to decide the satisfiability (resp. ω-satisfiability) of $\exists x_1 ... \exists x_n.\psi$, it is sufficient to do as follows: For each function $f : X \longrightarrow D \cup \{d'\}$ (there are exponentially many of them), decide the satisfiability (resp. ω-satisfiability) of the CTL formula ψ' over $AP \cup (T \times (D \cup \{d'\}))$, obtained by replacing each variable x_i by $f(x_i)$.

Since it is known that the satisfiability (resp. ω-satisfiability) of CTL formulae can be decided in exponential time, it follows that the satisfiability (resp. ω-satisfiability) problem of $\exists^*$-VCTL$_{pnf}$ is in EXPTIME.

The lower bound follows from the satisfiability problem of CTL. □

Theorem 5.16. *The model checking and ω-model checking problems of $\forall^*$-VCTL$_{pnf}$ are decidable in $EXPTIME$.*

Theorem 5.16 can be easily deduced from the following lemma.

Lemma 5.17. *Let $\mathcal{K} = (AP, X, S, R, S_0, I, L, L')$ be a VKS and $\forall x_1 ... \forall x_n.\psi$ be a $\forall^*$-VCTL$_{pnf}$ sentence. Then there is a computation tree (resp. ω-tree) $t = (Z, L)$ of $\mathcal{K}$ s.t. $t \models \exists x_1 ... \exists x_n.\overline{\psi}$ iff there is a computation tree (resp. ω-tree) $t' = (Z, L')$ of $\mathcal{K}$ s.t. $t' \models \exists x_1 ... \exists x_n.\overline{\psi}$ and t' contains at most $|X| + n$ different values.*

Proof. ($\Longrightarrow$) Suppose there is a computation tree (resp. ω-tree) $t = (Z, L)$ of $\mathcal{K}$ s.t. $t \models \exists x_1 ... \exists x_n.\overline{\psi}$. W.l.o.g., we assume that the number of different values used in t is greater than $|X| + n$ (otherwise we are done). Then there is an assignment $\lambda : \{x_1, \ldots, x_n\} \to \mathbb{D}$ s.t. $t \models_\lambda \overline{\psi}$. Let $D = \{\lambda(x_i) \mid 1 \le i \le n\}$ and $D' = \{d'_1, \ldots, d'_{|X|}\}$ be a set of $|X|$ data values that are different from all the data values occurring in t and the data values from $\mathbb{D}$.

Let $t' = (Z, L')$ be the computation tree (resp. ω-tree) obtained from t by replacing all the data values in $\mathbb{D} \setminus D$ with the data values in D', while respecting the invariants of the states in $\mathcal{K}$ as well as the reset constraints on the edges of $\mathcal{K}$. It is not hard to see that it is possible to do these replacements and these replacements do not affect the satisfaction of $\exists x_1 \ldots \exists x_n.\overline{\psi}$. Therefore, we obtain a computation tree (resp. ω-tree) t' of $\mathcal{K}$ that contains at most $|X| + n$ data values and satisfies $\exists x_1 ... \exists x_n.\overline{\psi}$.

($\Longleftarrow$) trivial. □

Proof of Theorem 5.16. Let $\mathcal{K} = (AP \cup T, X, S, R, S_0, I, L, L')$ be a VKS, and $\varphi = \forall x_1 ... \forall x_n \psi$ be a $\forall^*$-VCTL$_{pnf}$ sentence. By Lemma 5.17, we only need to consider the computation trees of $\mathcal{K}$ that contains at most $|X| + n$ data values. Let D be a set of data values of size $|X| + n$. Given a function $\lambda : X \longrightarrow D$ and a state $s \in S$, let $\Theta(s, \lambda)$ denote the set $(AP \cap L(s)) \cup \{(\tau, \lambda(x)) \mid (\tau, x) \in L(s)\}$.

Let k be the maximum number of successors of the states in $\mathcal{K}$. We construct a NTA (resp. NBTA) $\mathcal{A}_\mathcal{K} = (AP \cup T \times D, Q, \delta, Q_0, Q_f)$ over k-ary labeled trees s.t. $\mathcal{A}_\mathcal{K}$ defines the set of computation trees of $\mathcal{K}$ containing only data values from V.

- $Q = Q_f = \{(s, P) \in S \times 2^{AP \cup T \times D} \mid \exists \lambda : X \longrightarrow D : \lambda \models I(s) \wedge P = \Theta(s, \lambda)\}$,
- $Q_0 = Q \cap S_0 \times 2^{AP \cup T \times D}$,
- $\delta \subseteq Q \times 2^{AP \cup T \times D} \times Q^{\le k}$ is computed as follows: For every state $(s, P) \in Q$ s.t. $R(s) = \{s_1, ..., s_r\}$, δ contains all the transition rules $((s, P), P, ((s_1, P_1), \ldots, (s_r, P_r)))$ s.t. there are functions $\lambda, \lambda_1, ... \lambda_r : X \longrightarrow D$ satisfying that
  - $P = \Theta(s, \lambda)$,
  - for every $i : 1 \le i \le r$, $P_i = \Theta(s, \lambda_i)$, and for every $(reset, x) \notin L'(s, s_i)$: $\lambda(x) = \lambda_i(x)$.

Note that the size of $\mathcal{A}_\mathcal{K}$ is exponential over the size of $\mathcal{K}$ and φ.

On the other hand, construct a CTL formula φ' over $AP \cup T \times D$ from $\overline{\varphi}$ as the formula $\bigvee\limits_{\lambda:\{x_1,...,x_n\} \to D} \psi_\lambda$, where ψ_λ is obtained from $\overline{\psi}$ by replacing every occurrence of $\tau(x_i)$ (resp. $\neg\tau(x_i)$) s.t. $\tau \in T$ and $1 \le i \le n$ by $(\tau, \lambda(x_i))$ (resp. $\neg(\tau, \lambda(x_i))$).

To decide whether there is a computation tree (resp. ω-tree) of $\mathcal{K}$ satisfying $\overline{\varphi} = \exists x_1 \ldots \exists x_n.\overline{\psi}$, it remains to decide whether $\mathcal{L}(\mathcal{A}_\mathcal{K}) \cap \mathcal{L}(\varphi') \neq \emptyset$ (resp. $\mathcal{L}_\omega(\mathcal{A}_\mathcal{K}) \cap \mathcal{L}_\omega(\varphi') \neq \emptyset$).

Since it is well known that an equivalent NTA (resp. NBTA) of exponential size can be constructed from a CTL formula (cf. [58]), it follows that from each formula ψ_λ, a NTA (resp. NBTA) $\mathcal{A}_{\psi_\lambda}$ of exponential size can be constructed, then the union of these $\mathcal{A}_{\psi_\lambda}$, denoted by $\mathcal{A}'_\varphi$, is equivalent to φ' and still of exponential size. So the problem reduces to the nonemptiness of the intersection of $\mathcal{A}_\mathcal{K}$ and $\mathcal{A}_{\varphi'}$.

From Proposition 2.6, we conclude that the model checking problem (resp. ω-model checking problem) of $\forall^*$-VCTL$_{pnf}$ is in EXPTIME. □

6. Conclusion and future work

In this paper, we systematically investigated the theoretical aspects of VLTL and VCTL, the variable extensions of LTL and CTL respectively. At first, we compared the expressiveness of VLTL with the other logical formalisms over data words. Then we considered the decidability and complexity of the satisfiability and model checking problem of VLTL and VCTL, over both finite and infinite words (trees). We identified the decidability frontier of these decision problems of fragments of VLTL and VCTL and got a relatively complete picture (see Table 1).

For the future work, one obvious direction is to solve the questions left open in this paper. For instance, the questions whether the ω-satisfiability problems of $\exists^*\forall^*$-RVLTL$_{pnf}$ and EVCTL are decidable. It is also interesting to consider the model checking problem of VLTL and VCTL over counter machines or some proper extensions of pushdown systems that contain data values, e.g. pushdown register automata introduced in [59].

Acknowledgments

Fu Song is partially supported by Shanghai ChenGuang Program (No. 13CG21), and NSFC Projects (Nos. 61402179, and 61532019), the Open Project of Shanghai Key Laboratory of Trustworthy Computing (No. 07dz22304201404). Zhilin Wu is partially supported by the NSFC projects (Nos. 61100062, 61272135, 61472474, and 61572478).

References

[1] M.Y. Vardi, An automata-theoretic approach to linear temporal logic, in: 8th Banff Higher Order Workshop on Logics for Concurrency – Structure Versus Automata, 1995, pp. 238–266.
[2] E.A. Emerson, E.M. Clarke, Using branching time temporal logic to synthesize synchronization skeletons, Sci. Comput. Program. 2 (3) (1982) 241–266, http://dx.doi.org/10.1016/0167-6423(83)90017-5.
[3] Z. Manna, A. Pnueli, Verification of concurrent programs: the temporal framework, in: The Correctness Problem in Computer Science, Academic Press, 1981, pp. 215–273.
[4] A. Szalas, Concerning the semantic consequence relation in first-order temporal logic, Theor. Comput. Sci. 47 (1986) 329–334, http://dx.doi.org/10.1016/0304-3975(86)90157-X.
[5] A. Szalas, A complete axiomatic characterization of first-order temporal logic of linear time, Theor. Comput. Sci. 54 (2–3) (1987) 199–214, http://dx.doi.org/10.1016/0304-3975(87)90129-0.
[6] A. Szalas, L. Holenderski, Incompleteness of first-order temporal logic with until, Theor. Comput. Sci. 57 (2–3) (1988) 317–325, http://dx.doi.org/10.1016/0304-3975(88)90045-X.
[7] I.M. Hodkinson, F. Wolter, M. Zakharyaschev, Decidable fragment of first-order temporal logics, Ann. Pure Appl. Logic 106 (1–3) (2000) 85–134, http://dx.doi.org/10.1016/S0168-0072(00)00018-X.
[8] I.M. Hodkinson, F. Wolter, M. Zakharyaschev, Decidable and undecidable fragments of first-order branching temporal logics, in: 17th IEEE Symposium on Logic in Computer Science, LICS, Copenhagen, Denmark, 2002, pp. 393–402.
[9] V. Vianu, Automatic verification of database-driven systems: a new frontier, in: 12th International Conference on Database Theory, ICDT, St. Petersburg, Russia, 2009, pp. 1–13.
[10] E. Damaggio, A. Deutsch, R. Hull, V. Vianu, Automatic verification of data-centric business processes, in: 9th International Conference on Business Process Management, BPM, Clermont-Ferrand, France, 2011, pp. 3–16.
[11] S. Demri, R. Lazic, LTL with the freeze quantifier and register automata, ACM Trans. Comput. Log. 10 (3) (2009) 16, http://dx.doi.org/10.1145/1507244.1507246.
[12] M. Jurdzinski, R. Lazic, Alternation-free modal mu-calculus for data trees, in: Proceedings of the 22nd IEEE Symposium on Logic in Computer Science, LICS 2007, 10–12 July 2007, Wroclaw, Poland, 2007, pp. 131–140.
[13] D. Figueira, Alternating register automata on finite words and trees, Log. Methods Comput. Sci. 8 (1) (2012) 22, http://dx.doi.org/10.2168/LMCS-8(1:22)2012.
[14] A. Kara, T. Schwentick, T. Zeume, Temporal logics on words with multiple data values, in: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS, Chennai, India, 2010, pp. 481–492.
[15] S. Demri, D. Figueira, M. Praveen, Reasoning about data repetitions with counter systems, in: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, New Orleans, LA, USA, 2013, pp. 33–42.
[16] N. Decker, P. Habermehl, M. Leucker, D. Thoma, Ordered navigation on multi-attributed data words, in: 25th International Conference on Concurrency Theory, CONCUR, Rome, Italy, 2014, pp. 497–511.
[17] O. Grumberg, O. Kupferman, S. Sheinvald, Model checking systems and specifications with parameterized atomic propositions, in: 10th International Symposium on Automated Technology for Verification and Analysis, ATVA, Thiruvananthapuram, India, 2012, pp. 122–136.
[18] O. Grumberg, O. Kupferman, S. Sheinvald, An automata-theoretic approach to reasoning about parameterized systems and specifications, in: 11th International Symposium on Automated Technology for Verification and Analysis, ATVA, Hanoi, Vietnam, 2013, pp. 397–411.
[19] O. Grumberg, O. Kupferman, S. Sheinvald, A game-theoretic approach to simulation of data-parameterized systems, in: 12th International Symposium on Automated Technology for Verification and Analysis, ATVA, Sydney, NSW, Australia, 2014, pp. 348–363.
[20] R. Alur, P. Cerný, S. Weinstein, Algorithmic analysis of array-accessing programs, ACM Trans. Comput. Log. 13 (3) (2012) 27, http://dx.doi.org/10.1145/2287718.2287727.
[21] F. Song, Z. Wu, Extending temporal logics with data variable quantifications, in: 34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS, New Delhi, India, 2014, pp. 253–265.
[22] J. Bohn, W. Damm, O. Grumberg, H. Hungar, K. Laster, First-order-CTL model checking, in: 18th Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS, Chennai, India, 1998, pp. 283–294.
[23] I.M. Hodkinson, R. Kontchakov, A. Kurucz, F. Wolter, M. Zakharyaschev, On the computational complexity of decidable fragments of first-order linear temporal logics, in: 10th International Symposium on Temporal Representation and Reasoning/4th International Conference on Temporal Logic, TIME-ICTL, Cairns, Queensland, Australia, 2003, pp. 91–98.
[24] A. Degtyarev, M. Fisher, B. Konev, Monodic temporal resolution, ACM Trans. Comput. Log. 7 (1) (2006) 108–150, http://dx.doi.org/10.1145/1119439.1119443.

[25] U. Hustadt, B. Konev, A. Riazanov, A. Voronkov, Temp: a temporal monodic prover, in: 2nd International Joint Conference on Automated Reasoning, IJCAR, Cork, Ireland, 2004, pp. 326–330.
[26] B. Konev, A. Degtyarev, C. Dixon, M. Fisher, U. Hustadt, Mechanising first-order temporal resolution, Inf. Comput. 199 (1–2) (2005) 55–86, http://dx.doi.org/10.1016/j.ic.2004.10.005.
[27] C. Dixon, M. Fisher, B. Konev, Tractable temporal reasoning, in: 20th International Joint Conference on Artificial Intelligence, IJCAI, Hyderabad, India, 2007, pp. 318–323.
[28] S. Demri, D. D'Souza, An automata-theoretic approach to constraint LTL, Inf. Comput. 205 (3) (2007) 380–415, http://dx.doi.org/10.1016/j.ic.2006.09.006.
[29] F. Song, T. Touili, Pushdown model checking for malware detection, in: 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS, Tallinn, Estonia, 2012, pp. 110–125.
[30] F. Song, T. Touili, LTL model-checking for malware detection, in: 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, Rome, Italy, 2013, pp. 416–431.
[31] F. Song, T. Touili, Pushdown model checking for malware detection, Int. J. Softw. Tools Technol. Transf. 16 (2) (2014) 147–173, http://dx.doi.org/10.1007/s10009-013-0290-1.
[32] M.C. Browne, E.M. Clarke, O. Grumberg, Reasoning about networks with many identical finite state processes, Inf. Comput. 81 (1) (1989) 13–31, http://dx.doi.org/10.1016/0890-5401(89)90026-6.
[33] E.A. Emerson, J. Srinivasan, A decidable temporal logic to reason about many processes, in: 9th Annual ACM Symposium on Principles of Distributed Computing, PODC, Quebec City, Quebec, Canada, 1990, pp. 233–246.
[34] A.P. Sistla, S.M. German, Reasoning with many processes, in: Proceedings of the Symposium on Logic in Computer Science, LICS'87, June 22–25, 1987, Ithaca, New York, USA, 1987, pp. 138–152.
[35] S.M. German, A.P. Sistla, Reasoning about systems with many processes, J. ACM 39 (3) (1992) 675–735, http://dx.doi.org/10.1145/146637.146681.
[36] E.A. Emerson, V. Kahlon, Reducing model checking of the many to the few, in: 17th International Conference on Automated Deduction, CADE, Pittsburgh, PA, USA, 2000, pp. 236–254.
[37] E.A. Emerson, K.S. Namjoshi, On reasoning about rings, Int. J. Found. Comput. Sci. 14 (4) (2003) 527–550, http://dx.doi.org/10.1142/S0129054103001881.
[38] M. Kaminski, N. Francez, Finite-memory automata, Theor. Comput. Sci. 134 (2) (1994) 329–363, http://dx.doi.org/10.1016/0304-3975(94)90242-9.
[39] O. Grumberg, O. Kupferman, S. Sheinvald, Variable automata over infinite alphabets, in: 4th International Conference on Language and Automata Theory and Applications, LATA, Trier, Germany, 2010, pp. 561–572.
[40] F. Neven, T. Schwentick, V. Vianu, Towards regular languages over infinite alphabets, in: 26th International Symposium on Mathematical Foundations of Computer Science, MFCS, Marianske Lazne, Czech Republic, 2001, pp. 560–572.
[41] M. Kaminski, D. Zeitlin, Finite-memory automata with non-deterministic reassignment, Int. J. Found. Comput. Sci. 21 (5) (2010) 741–760, http://dx.doi.org/10.1142/S0129054110007532.
[42] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, C. David, Two-variable logic on words with data, in: 21th IEEE Symposium on Logic in Computer Science, LICS, Seattle, WA, USA, 2006, pp. 7–16.
[43] A. Kara, T. Schwentick, T. Tan, Feasible automata for two-variable logic with successor on data words, in: 6th International Conference on Language and Automata Theory and Applications, LATA, A Coruña, Spain, 2012, pp. 351–362.
[44] Z. Wu, Commutative data automata, in: 26th International Workshop, 21st Annual Conference of the EACSL on Computer Science Logic, CSL, Fontainebleau, France, 2012, pp. 528–542.
[45] M. Bojanczyk, S. Lasota, An extension of data automata that captures xpath, Log. Methods Comput. Sci. 8 (1) (2012) 5, http://dx.doi.org/10.2168/LMCS-8(1:5)2012.
[46] Z. Wu, A decidable extension of data automata, in: 2nd International Symposium on Games, Automata, Logics and Formal Verification, GandALF, Minori, Italy, 2011, pp. 116–130.
[47] A. Manuel, R. Ramanujam, Class counting automata on datawords, Int. J. Found. Comput. Sci. 22 (4) (2011) 863–882, http://dx.doi.org/10.1142/S0129054111008465.
[48] T. Tan, Extending two-variable logic on data trees with order on data values and its automata, ACM Trans. Comput. Log. 15 (1) (2014) 8, http://dx.doi.org/10.1145/2559945.
[49] T. Tan, On pebble automata for data languages with decidable emptiness problem, J. Comput. Syst. Sci. 76 (8) (2010) 778–791, http://dx.doi.org/10.1016/j.jcss.2010.03.004.
[50] T. Tan, Graph reachability and pebble automata over infinite alphabets, ACM Trans. Comput. Log. 14 (3) (2013) 19, http://dx.doi.org/10.1145/2499937.2499940.
[51] O. Kupferman, Variations on safety, in: 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, Grenoble, France, 2014, pp. 1–14.
[52] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata techniques and applications, available on http://www.grappa.univ-lille3.fr/tata, 2007.
[53] E. Grädel, W. Thomas, T. Wilke (Eds.), Automata, Logics, and Infinite Games: A Guide to Current Research, Lecture Notes in Computer Science, vol. 2500, Springer, 2002.
[54] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin, Two-variable logic on data words, ACM Trans. Comput. Log. 12 (4) (2011) 27, http://dx.doi.org/10.1145/1970398.1970403.
[55] N. Decker, D. Thoma, On freeze LTL with ordered attributes, CoRR, arXiv:1504.06355.
[56] H. Björklund, T. Schwentick, On notions of regularity for data languages, Theor. Comput. Sci. 411 (4–5) (2010) 702–715, http://dx.doi.org/10.1016/j.tcs.2009.10.009.
[57] T.A. Henzinger, O. Kupferman, R. Majumdar, On the universal and existential fragments of the mu-calculus, Theor. Comput. Sci. 354 (2) (2006) 173–186, http://dx.doi.org/10.1016/j.tcs.2005.11.015.
[58] O. Kupferman, M.Y. Vardi, P. Wolper, An automata-theoretic approach to branching-time model checking, J. ACM 47 (2) (2000) 312–360, http://dx.doi.org/10.1145/333979.333987.
[59] E.Y.C. Cheng, M. Kaminski, Context-free languages over infinite alphabets, Acta Inform. 35 (3) (1998) 245–267.