Temporal Logics on Words with Multiple Data Values*

Ahmet Kara[1], Thomas Schwentick[1], and Thomas Zeume[1]

1 **TU Dortmund**
Germany
{ahmet.kara, thomas.schwentick, thomas.zeume}@cs.tu-dortmund.de

Abstract

The paper proposes and studies temporal logics for attributed words, that is, data words with a (finite) set of (attribute,value)-pairs at each position. It considers a basic logic which is a semantical fragment of the logic $\text{LTL}_1^{\downarrow}$ of Demri and Lazic with operators for navigation into the future and the past. By reduction to the emptiness problem for data automata it is shown that this basic logic is decidable. Whereas the basic logic only allows navigation to positions where a fixed data value occurs, extensions are studied that also allow navigation to positions with different data values. Besides some undecidable results it is shown that the extension by a certain UNTIL-operator with an inequality target condition remains decidable.

1998 ACM Subject Classification F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic – Temporal logic; F.4.3 [Mathematical Logic and Formal Languages]: Formal Languages – Decision problems

Keywords and phrases Expressiveness, Decidability, Data words

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.481

1 Introduction

Motivated by questions from XML theory and automated verification, extensions of (finite or infinite) strings by data values from unbounded domains have been studied intensely in recent years. Various logics and automata for such data words have been invented and investigated.

A very early study by Kaminski and Francez [16] considered automata on strings over an "infinite alphabet". In [7], *data words* were invented as finite sequences of pairs (σ, d), where σ is a symbol from a finite alphabet and d a value from a possibly infinite domain. In [6] multi-dimensional data words were considered where every position carries N variable valuations, for some fixed N. Similar models can be found for instance in [2] and other work on parameterized verification. More powerful models were investigated in [19] and [14] where every position is labeled by the state of a relational database, i.e., by a set of relations over a fixed signature.

For the basic model of data strings with one data value per position a couple of automata models and logics have been invented and their algorithmic and expressive properties have been studied. On the automata side we mention register automata [16, 8, 22] (named *finite memory automata* in [16, 8]), pebble automata [22, 24], alternating 1-register automata [12], data automata [5] (or the equivalent class memory automata [3]).

* We acknowledge the financial support by the German DFG under grant SCHW 678/4-1.

On the logical side, classical logics like two-variable first-order logic [5] have been studied and recently order comparisons between data values have been considered [20, 23]. The satisfiability problem for two-variable first-order logic over data words is decidable if data values can only be compared for equality but positions can be compared with respect to the linear order and the successor relation [5]. However, the complexity is unknown. It is elementary if and only if testing reachability in Petri nets is elementary as well [5]. The proof of decidability uses *data automata*, a strong automata model with decidable non-emptiness.

More relevant for this paper are previous investigations of temporal logics on data words. A pioneering contribution was by Demri and Lazic [12] (the journal version of [11]) which introduced Freeze LTL. In a nutshell, Freeze LTL extends LTL by *freeze quantifiers* which[1] allow to "store" the current data value in a register and to test at a possibly different position whether that position carries the same value. Freeze LTL has a decidable finite satisfiability problem if it is restricted to one register ($\mathrm{LTL}_1^{\downarrow}$) and to future navigation, but the complexity is not primitive recursive. With one register and past (and future) navigation it is undecidable. In [15] it is shown that these lower bounds even hold if only navigation with F and P (but without X) are allowed.

In [12], also a restriction of $\mathrm{LTL}_1^{\downarrow}$, *simple* $\mathit{LTL}_1^{\downarrow}$, was investigated and it was shown that it is expressively equivalent to two-variable logics. The restriction requires that (syntactically) between each value test and the corresponding freeze quantifier there is at most one temporal operator and it disallows Until and Since navigation but allows past navigation. Thanks to the (effective) equivalence to two-variable logics, simple $\mathrm{LTL}_1^{\downarrow}$ is decidable.

One of our aims in this paper was to find a decidable temporal logic on data words with past navigation that is more expressive than simple $\mathrm{LTL}_1^{\downarrow}$. In particular it should allow Until navigation with reference to data values. On the other hand, the logics we study are semantical fragments of $\mathrm{LTL}_1^{\downarrow}$. Furthermore this work was motivated by the decidable logic $\mathrm{CLTL}^{\diamond}$ for multi-attribute data words [10]. It allows to test whether somewhere in the future (or past) a current data value occurs and it can compare data values between two positions of bounded distance. The logics proposed in this paper are intended to have more expressive power than $\mathrm{CLTL}^{\diamond}$ while retaining its decidability.

Contribution

We propose and investigate temporal logics for multi-attribute data words. An attributed word is a string which can have a finite number of (attribute,value)-pairs at each position (in the spirit of XML) and has propositions rather than symbols (in the spirit of LTL).

We first define Basic Data LTL which mimics the navigation abilities of simple $\mathrm{LTL}_1^{\downarrow}$, if only positive register tests are used. As sequences of such navigation steps do not do any harm we drop the requirement to freeze the data value at every step and replace freeze quantifiers by a class quantifier which restricts a sub-formula to the positions at which this data value appears. We show that a slight extension of this logic captures simple $\mathrm{LTL}_1^{\downarrow}$ (Proposition 2) and that it is decidable (Theorem 1). Although strictly more expressive than $\mathrm{CLTL}^{\diamond}$, the decidability proof for Basic Data LTL is conceptually simpler than the proof given in [10]. It uses an encoding of multi-attribute words by data words and a reduction to non-emptiness of data automata. A similar multi-attribute encoding has already been used in [13]. The result generalizes to attributed ω-words (Theorem 3). Some obvious extensions (by navigation with respect to two data values or Until navigation where intermediate positions

[1] We note that the freeze quantifier itself was used already in [9] and in previous work, e.g., in [1].

can be tested by data-free formulas) are undecidable (Theorems 4 and 6, respectively).

Finally, we add a powerful Until-operator to Basic Data LTL, which allows to navigate to a position with a data value that is *different* from the value of a given attribute at the starting position. Furthermore, it can test properties of intermediate positions by arbitrary sub-formulas and can even test (in a limited way) whether intermediate positions have attribute values different from or equal to the value on the starting position. The resulting logic can express all properties expessible in two-variable first-order logic and contains the Until operator. That this logic is still decidable is the main technical contribution of the paper.

The paper is organized as follows. In Section 2, we define attributed words and Basic Data LTL and give some example properties. In Section 3, we compare Basic Data LTL with other logics. Section 4 shows that Basic Data LTL is decidable and presents undecidability results for some extensions. Section 5 introduces the extended Until operator and shows decidability of the resulting logic. It also shows (the simple fact) that an Until-operator that navigates with respect to equality and allows (only) data-free intermediate tests quickly leads to an undecidable logic. We conclude in Section 6. Due to lack of space most proofs are only sketched or even missing. They can be found in the full version of the paper [17].

Related work

We discussed many related papers above. Another approach, combining temporal and classical logics, was studied in [14]. It allows to navigate by temporal operators and to evaluate first-order formulas in states. Properties depending on values at different states can be stated by global universal quantification of values. In [6] a first-order logic on multi-dimensional data words was studied.

Acknowledgements

The idea to extend the temporal logic that is equivalent to two-variable logics by Until operators (without reference to data) goes back to a suggestion by Mikołaj Bojanczyk [4]. We are also indebted to Volker Weber with whom we carried out first investigations before he tragically passed away in 2009. The remarks by the reviewers of FSTTCS 2010 helped to improve the presentation and to add some additional references.

2 Definitions

We first fix the data model and define BD-LTL afterwards. Finally we give an example that illustrates the way in which properties can be expressed

2.1 Attributed words

Let $\mathcal{PROP}$ and $\mathcal{ATT}$ be (possibly infinite) sets of propositions and attributes and $\mathcal{D}$ an infinite set of data values. An *attributed word* w is a finite word where every position carries a finite set $\{p_1, \dots, p_l\}$ of propositions from $\mathcal{PROP}$ and a finite set $\{(a_1, d_1), \dots, (a_k, d_k) \mid a_i \neq a_j \text{ for } i \neq j\}$ of attribute-value pairs from $\mathcal{ATT} \times \mathcal{D}$.

Given an attributed word w we denote the proposition set of position i in w by $w[i].\mathcal{P}$. A position i is a *p-position* if $p \in w[i].\mathcal{P}$. By $w[i].@a$ we denote the value of attribute a on position i. If position i does not carry attribute a, then $w[i].@a = nil \notin D$. The *word projection* of an attributed word $w = w_1 \dots w_n$ is defined by $\mathit{str}(w) := w[1].\mathcal{P} \dots w[n].\mathcal{P}$. By $\mathit{pos}_d(w)$ we denote the set of *class positions* of d in w, that is, the set of positions of w with at least one attribute with value d. The *class word* $\mathit{class}_d(w)$ of w with respect to d is the restriction of w to the positions of $\mathit{pos}_d(w)$.

We always consider sets of words over some finite set $\mathcal{P}$ of propositions and a finite set $\mathcal{V}$ of attributes[2]. We call an attributed word w *$\mathcal{V}$-complete* for a finite set $\mathcal{V} \subseteq \mathcal{ATT}$ if every position of w has exactly one pair (a, d_a) for each $a \in \mathcal{V}$. A $\{a\}$-complete word is called *1-attributed word* . We refer to the value of attribute $@a$ at a position i in a 1-attributed word as *the data value of* i. There is an immediate correspondence between data strings (that is, sequences of (symbol,value) pairs) and 1-attributed words. Thus, we use in this paper automata and logics that were introduced for data strings also for 1-attributed words.

Attributed ω-words are defined accordingly.

For $i, j \in \mathbb{N}$ with $i \leq j$ we denote the interval $\{i, i+1, \ldots j\}$ by $[i, j]$. As usual we use round brackets to denote open intervals, e.g., $[3, 5) = \{3, 4\}$.

2.2 Basic Data LTL

The logic Basic Data LTL (abbreviated: BD-LTL) has two main types of formulas, *position formulas* and *class formulas*, where, intuitively, class formulas express properties of class words. We first state the syntax of the logic and give an intuitive explanation of its non-standard features afterwards.

We fix a finite set $\mathcal{P} \subseteq \mathcal{PROP}$ of propositions and a finite set $\mathcal{V} \subseteq \mathcal{ATT}$ of attributes.

The syntax of *position formulas* φ and *class formulas* ψ of BD-LTL (over $\mathcal{P}$ and $\mathcal{V}$) are defined as follows.

$$\begin{aligned} \varphi &::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathrm{X}\varphi \mid \mathrm{Y}\varphi \mid \varphi\mathrm{U}\varphi \mid \varphi\mathrm{S}\varphi \mid \mathrm{C}^{\delta}_{@a}\psi \\ \psi &::= \varphi \mid @a \mid \neg\psi \mid \psi \vee \psi \mid \mathrm{X}^{=}\psi \mid \mathrm{Y}^{=}\psi \mid \psi\,\mathrm{U}^{=}\,\psi \mid \psi\mathrm{S}^{=}\psi \end{aligned}$$

Here, $p \in \mathcal{P}$, $a \in \mathcal{V}$, $\delta \in \mathbb{Z}$. Intuitively, the quantifier $\mathrm{C}_{@a}\psi$ restricts the evaluation of ψ to the class word induced by attribute a at the current position.

Next we define the formal semantics of position formulas. Let w be an attributed word and i a position on w:

- $w, i \models p$ if $p \in w[i].\mathcal{P}$;
- $w, i \models \neg\varphi$ if $w, i \not\models \varphi$;
- $w, i \models \varphi_1 \vee \varphi_2$ if $w, i \models \varphi_1$ or $w, i \models \varphi_2$;
- $w, i \models \mathrm{X}\varphi$ if $i + 1 \leq |w|$ and $w, i + 1 \models \varphi$;
- $w, i \models \varphi_1\mathrm{U}\varphi_2$ if there exists a $j \geq i$ such that $w, j \models \varphi_2$ and $w, j' \models \varphi_1$ for all $j' \in [i, j)$;
- $w, i \models \mathrm{C}^{\delta}_{@a}\psi$ if $w[i].@a \neq \mathit{nil}$, $i + \delta \in [1, |w|]$, and $w, i + \delta, w[i].@a \models \psi$.

The operators Y and S are the past counterparts of X and U respectively. Their semantics is defined analogously[3].

Next, we define the semantics of class formulas. Let w be an attributed word, i a position on w and d a data value.

- $w, i, d \models \varphi$ if $w, i \models \varphi$;
- $w, i, d \models @a$ if $w[i].@a = d$;
- $w, i, d \models \mathrm{X}^{=}\varphi$ if there exists a $j \in \mathit{pos}_d(w)$ with $j > i$, and for the smallest such j it holds $w, j, d \models \varphi$;

[2] As we will use $\mathcal{A}$ for automata we use $\mathcal{V}$ here: **V**ariables.
[3] To avoid ambiguity: $p\mathrm{S}q$ holds if there is a q-position in the past and at the intermediate positions p holds.

- $w, i, d \models \varphi_1 \, \mathrm{U}^{=} \varphi_2$ if there exists a $j \in pos_d(w)$ with $j \geq i$ such that $w, j, d \models \varphi_2$ and $w, k, d \models \varphi_1$ for all $k \in pos_d(w) \cap [i, j)$.

For the past class operators Y and S the semantics is defined analogously and the semantics of the Boolean connectors is as usual. Finally, $w \models \varphi$, if $w, 1 \models \varphi$. We denote the set of positional formulas by BD-LTL.

Besides $\bot$ and $\top$ we use the following usual abbreviations:

$$\mathrm{F}\varphi := \top \mathrm{U} \varphi \quad \mathrm{G}\varphi := \neg\mathrm{F}\neg\varphi \quad \mathrm{P}\varphi := \top\mathrm{S}\varphi \quad \mathrm{H}\varphi := \neg\mathrm{P}\neg\varphi$$

The abbreviations $\mathrm{F}^{=}$ and $\mathrm{G}^{=}$ and their past counterparts are defined analogously. Furthermore, we abbreviate $\mathrm{C}^{\delta}_{@a}@b$ by $@a = \mathrm{X}^{\delta}@b$.

2.3 Example: a simple client/server scenario

The following example illustrates how properties can be expressed in BD-LTL.

Consider an internet platform that uses m servers $S_1, \ldots, S_m$ to process queries from clients. Every client shall have a unique client number. As we do not know beforehand how many clients will use the platform, we model the client numbers by the set $\mathcal{D} = \mathbb{N}$.

Each of the servers can either idle, be queried by a client or serve the answer for a query. For server j, the actions are modeled by the set of propositions $\{q_j, s_j, i_j\}$. Runs of the internet platform can now be represented by an attributed word with attribute set $\mathcal{ATT} = \{S_1, \ldots, S_m\}$ and set of propositions $\bigcup_{1 \leq j \leq m} \{q_j, s_j, i_j\}$. That a server S_j shall perform exactly one action from $\{q_j, s_j, i_j\}$ at any given time, can be easily expressed by a BD-LTL-formula.

Let us look at an example system with three servers A, B and C. An example run represented as an attributed word could look as follows.

Pos	1	2	3	4	5	6
Props	$\{q_A, q_B, i_C\}$	$\{q_A, q_B, q_C\}$	$\{s_A, q_B, s_C\}$	$\{s_A, s_B, i_C\}$	$\{i_A, s_B, q_C\}$	$\{i_A, s_B, s_C\}$
A	1	2	2	1	–	–
B	2	3	4	2	3	4
C	–	1	1	–	2	2

Here, e.g., at position 5 server A is idling, server B is serving client 3 and server C is queried by client 2. Properties of runs can be expressed by BD-LTL formulas:

- Queries are always served and a client can query a second time on a server only after the previous query has been served:

$$\bigwedge_{Z \in \{A,B,C\}} \mathrm{G}(q_Z \to \mathrm{C}_{@Z}(\mathrm{X}^{=}(@Z \to \neg q_Z) \, \mathrm{U}^{=}(@Z \wedge s_Z)))$$

- A server Z can serve a client only if there is an unanswered query by that client (i.e. the last action by that client on Z was a query):

$$\bigwedge_{Z \in \{A,B,C\}} \mathrm{G}(s_Z \to \mathrm{C}_{@Z}(\mathrm{Y}^{=}(\neg @Z)\mathrm{S}^{=}(@Z \wedge q_Z))))$$

- A client with an open query on server A shall only be allowed to query server C until server A answered the query:

$$G(q_A \to \mathrm{C}_{@A}(\neg @B \wedge \mathrm{X}^{=}((\neg(q_A \wedge A) \wedge \neg(q_B \wedge B)) \, \mathrm{U}^{=} s_A)))$$

3 Expressiveness of BD-LTL

In this section we will give a short overview of established logics on strings with data values and outline how BD-LTL fits in. We give a short introduction to freeze LTL and CLTL$^\diamond$, see [12] and [10] for more details. Afterwards we compare these two logics to BD-LTL.

3.1 BD-LTL versus LTL$_1^\downarrow$

Freeze LTL is an extension of LTL for data words by a freeze quantifier that binds the data value of the current position to a variable (aka register) and allows to compare the value of a position with the value bound to a variable. Satisfiability for freeze LTL is undecidable even for two registers [12], therefore [12] proposed the 1-register fragment LTL$_1^\downarrow$. In the framework of 1-attributed words, formulas of LTL$_1^\downarrow$ are of the form

$$\varphi ::= p \mid {\downarrow}\varphi \mid {\uparrow} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathrm{X}\varphi \mid \mathrm{Y}\varphi \mid \varphi\mathrm{U}\varphi \mid \varphi\mathrm{S}\varphi.$$

The formal semantics of LTL$_1^\downarrow$ (on data strings) can be found in [12]. We illustrate it by a simple example: the formula $\mathrm{G}(p \to {\downarrow}\mathrm{F}(q \wedge {\uparrow}))$ expresses that each p-position has a future q-position with the same data value.

In [12], the fragment simple LTL$_1^\downarrow$ was invented, where at most one temporal operator is allowed between the the freeze quantifier $\downarrow$ and a value test $\uparrow$. Furthermore, only the unary temporal operators $\mathrm{X}^k, \mathrm{Y}^k, \mathrm{X}^k\mathrm{F}, \mathrm{Y}^k\mathrm{P}$, $k \in \mathbb{N}$ are allowed. Here, $\mathrm{X}^k\mathrm{F}$ is considered a single operator, that is ${\downarrow}\mathrm{X}^k\mathrm{F}{\uparrow}$ is an allowed formula. The relative expressive power of BD-LTL and LTL$_1^\downarrow$ can be summarized in the following two propositions.

▶ Proposition 1. Every property of 1-attributed words that is expressible in BD-LTL can also be expressed in LTL$_1^\downarrow$.

The statement also holds for all extensions of BD-LTL considered in Section 5. Note however, that LTL$_1^\downarrow$ is undecidable whereas BD-LTL and its main extension in Section 5 are decidable.

▶ Proposition 2. The following logics are equivalent on 1-attributed words
(i) Simple LTL$_1^\downarrow$
(ii) BD-LTL without Until and Since extended by $\mathrm{F}^\delta_{\neq}$ and $\mathrm{P}^\delta_{\neq}$.
Here, $\mathrm{F}^\delta_{\neq}\varphi$ intuitively navigates to a future position of distance $\geq \delta$ with a different data value and evaluates φ there. In the notation of Section 5 it is an abbreviation for $\top\mathrm{U}^\delta_{@a}(\overline{@a} \wedge \varphi)$. Note, that an analogous operator $F^\delta_{=}\varphi$ for equal data values can be simulated by $\mathrm{C}^\delta_{@a}F^{=}\varphi$. The proof of both propositions is straightforward and therefore omitted.

3.2 BD-LTL versus CLTL$^\diamond$

Temporal logic of repeating values (CLTL$^\diamond$) was introduced in [10]. CLTL$^\diamond$-formulas are of the form $\varphi ::= x = X^\delta y \mid x = \diamond y \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathrm{X}\varphi \mid \varphi\mathrm{U}\varphi \mid \mathrm{Y}\varphi \mid \varphi\mathrm{S}\varphi$, where x, y are from a set of variables. A CLTL$^\diamond$-formula with variables $\{x_1, \dots, x_m\}$ is evaluated on sequences of m-tuples of data values (without labels from a finite set) but the extension to $\{x_1, \dots, x_m\}$-complete attributed strings is straightforward. A formula $x = X^\delta y$ tests whether component x of the current position has the same data value as component y of the δ-next position. A formula $x = \diamond y$ is true if there is a (strict) future position with the same data value on component y as the current position has on component x. The semantics of all other operators is as usual. The following proposition is straightforward, since $x = \diamond y$ and $x = X^\delta y$ can be encoded by $\mathrm{C}^0_{@x}\mathrm{X}^{=}\mathrm{F}^{=}@y$ and $\mathrm{C}^\delta_{@x}@y$, respectively.

▶ Proposition 3. On $\{x_1, \dots, x_m\}$-complete attributed words BD-LTL is strictly more expressive than CLTL$^\diamond$.

4 Decidability of Basic Data LTL

This section states the main decidability result for BD-LTL and undecidability results for some of its extensions.

▶ **Theorem 1.** *Satisfiability for BD-LTL is decidable.*

The proof of this result proceeds in two main steps. First it is shown that the satisfiability problem for arbitrary attributed words can be reduced to the case of 1-attributed words. A similar reduction from the multi-attribute to the 1-attribute case (for a different logic) has been given in [13]. For 1-attributed words, BD-LTL-formulas can be translated into data automata [5] and thus the satisfiability problem for BD-LTL can be reduced to the decidable non-emptiness problem for data automata.

In a nutshell, a *data automaton* $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ consists of a finite state transducer $\mathcal{B}$ (the *base automaton*) and a finite state automaton $\mathcal{C}$ (the *class automaton*). The string projection of a given 1-attributed word w is processed by the base automaton, firstly. Then the output w' of $\mathcal{B}$ is processed class-wise by the class automaton, i.e. $\mathcal{C}$ is run for every data value d on the class word $class_d(w)$. $\mathcal{A}$ accepts w, if $\mathcal{B}$ accepts and $\mathcal{C}$ accepts all class words.

We give only a proof sketch, see the full version of this article for a detailed proof [17].

▶ **Theorem 2.** *Satisfiability for BD-LTL on 1-attributed words is decidable.*

Proof. (Sketch.)

Let φ be a BD-LTL formula over a proposition set $\mathcal{P}$ and the attribute set $\{a\}$.

In the following we often call 1-attributed words simply *words*. Our automata will expect instead of words w over P extended words w' with additional propositions. First, w' allows the subformulas of φ as propositions. The intention is that a position i of w' is marked with ψ if and only if $w, i \models \psi$. Furthermore, we use propositions $=_r$ for every $r \in \{-N, \ldots, -1, 1, \ldots, N\}$, for some N that is at least as large as every δ occurring in φ. Proposition $=_r$ shall hold at position i if and only if $w[i].@a = w[i+r].@a$.

The data automaton $\mathcal{A}$ now checks whether those additional propositions are correct. $\mathcal{A}$ is the intersection of data automata for the following conditions:

i) The propositions $=_r$ are placed correctly.
ii) Subformulas are placed correctly (i.e. position i is labeled with proposition ψ if and only if ψ is fulfilled on position i).
iii) φ is placed on the first position

Condition iii) can be easily checked. Condition i) can be checked by a data automaton [3].

For ii), a data automaton for every subformula ψ is constructed, assuming the correctness of subformulas of ψ. Checking the correctness is straightforward for subformulas ψ of type p, $\neg\chi$, $\chi \vee \chi$, $\mathrm{X}\chi$, $\mathrm{Y}\chi$, $\chi\mathrm{U}\chi$, $\chi\mathrm{S}\chi$. Basically, these formulas can be checked solely by the base automaton. The construction is equally straightforward for all types of class formulas. In these cases, basically only class automata are needed.

To deal with the δ-shift in formulas of the form $\mathrm{C}^{\delta}_{@a}\psi$ we use the propositions $=_r$. E.g., to validate propositions of the form $\psi = \mathrm{C}^{7}_{@a}\mathrm{F}^{=}\chi$ at position i, the class automaton $\mathcal{A}_\psi$ infers from the $=_r$ propositions how many positions the class word has between i and $i+7$, then it skips these positions and starts searching for a χ-position from there. ◀

Theorem 1 can be easily extended to the case of attributed attributed ω-words as in [5].

▶ **Theorem 3.** *Satisfiability for BD-LTL on attributed ω-words is decidable.*

Extensions of BD-LTL quickly yield undecidability. We consider two such extensions here.

BD-LTL with Navigation along Tuples. We extend $\mathrm{C}_{@a}$ to a quantifier $\mathrm{C}_{@a,@b}$ that 'freezes' the values d_a and d_b of the attributes a and b, respectively. Operators $\mathrm{X}^=, \mathrm{Y}^=, \mathrm{U}^=$ and $\mathrm{S}^=$ in the scope of $\mathrm{C}_{@a,@b}$ then move along positions that have attributes with data values d_a and d_b. At such positions the values of tuples of attributes can be tested for equality with (d_a, d_b). For example the property 'there is a future position with proposition p where attribute c carries the same data value as attribute a at the current position, likewise for d and b' can be expressed by $\mathrm{C}_{@a,@b}F^=((@c, @d) \wedge p)$.

However, already a restricted version of this extension is undecidable. We consider the operators $\mathrm{X}_{@\mathrm{a},@\mathrm{b}}$ and $\mathrm{Y}_{@\mathrm{a},@\mathrm{b}}$. Let the semantics of $\mathrm{X}_{@\mathrm{a},@\mathrm{b}}$ be defined by $w, i \models \mathrm{X}_{@\mathrm{a},@\mathrm{b}}\varphi$ if there is a $j > i$ with $w[i].@a = w[j].@a$ and $w[i].@b = w[j].@b$ and for the smallest such j it holds $w, j \models \varphi$. The operator $\mathrm{Y}_{@a,@b}$ is defined analogously.

▶ **Theorem 4.** *BD-LTL extended by the operators $\mathrm{X}_{@a,@b}$ and $\mathrm{Y}_{@a,@b}$ is undecidable on finite (or infinite) attributed words.*

The proof is along the lines of Proposition 27 in [5] by a reduction from the Post Correspondence Problem (PCP).

BD-LTL with From-Now-On Operator. The *from-now-on*-operator N introduced in [18] restricts the range of past operators. For an attributed word $w = w_1 \dots w_n$ and a position i of w let $suf_i(w) := w_i \dots w_n$ be the suffix of w starting at position i. The semantics of N is then defined by

- $w, i \models \mathrm{N}\varphi$ if $suf_i(w), 1 \models \varphi$

▶ **Theorem 5.** *BD-LTL extended by the operator* N *is undecidable on finite (or infinite) attributed words.*

The proof is by a reduction from the non-emptiness problem for Minsky two counter automata [21].

5 Extended Navigation

As already discussed before, the navigational abilities of BD-LTL are limited. It seemingly cannot[4] even express the simple property that for every p-position i there is a q-position $j > i$ such that $w[j].@b \neq w[i].@a$. Furthermore, in class formulas $\rho\mathrm{U}^=\tau$, the formula ρ can only refer to positions of the current class. Of course, it would be desirable to allow more general forms of "Until navigation".

In this section we discuss different possibilities to extend the navigational abilities of BD-LTL in an "Until fashion", some of which are decidable and some undecidable. In particular, we exhibit an U-operator with the ability to navigate to a position with a different attribute value and to state some properties on (all) intermediate positions and show that BD-LTL remains decidable with this extension. The property stated in the previous paragraph can be expressed using this operator.

The extensions we study allow formulas of the type $\rho\mathrm{U}^{\delta}_{@a}\tau$, where $\delta \geq 0$. Intuitively, this operator "freezes" the current value of attribute a and searches for a position j such that τ

[4] We did not attempt to find a proof for this statement as we were aiming for an extended logic, anyway. However, we did not find a simple way to express the property.

holds at j and ρ hold everywhere in $[i+\delta, j)$. In formulas as above, we will refer to ρ as the *intermediate formula* and τ as the *target formula*. The "shift" parameter δ is needed as we aim to design a semantic extension of simple LTL$^{\downarrow}_{1}$.

Syntactically, the formulas ρ and τ are positive Boolean combinations of position formulas and positive and negative attribute tests. More formally, we define the syntax of *U-subformulas* χ by $\chi ::= \varphi \mid @b \mid \overline{@b} \mid \chi \vee \chi \mid \chi \wedge \chi$. Intuitively, negative attribute tests $\overline{@b}$ check that attribute b has a value (!) that is different from the current frozen value.

Thus, the semantics of formulas $\rho \mathrm{U}^{\delta}_{@a}\tau$, where ρ and τ are U-subformulas, is defined by the following additional rules.

- $w, i \models \rho \mathrm{U}^{\delta}_{@a}\tau$ if there exists a $j \geq i+\delta$ such that $w, j, w[i].@a \models \tau$ and $w, k, w[i].@a \models \rho$ for all $k \in [i+\delta, j)$
- $w, i, d \models \overline{@b}$ if $w[i].@b \notin \{nil, d\}$.

We simply use $\mathrm{U}_{@a}$ instead of $\mathrm{U}^{0}_{@a}$. We remark that $\rho \mathrm{U}^{-\delta}_{@a}\tau$, for $\delta \geq 0$ can be expressed by $(\rho\, \mathrm{U}_{@a}\, \tau \wedge \bigwedge_{i=1}^{\delta} \rho_i) \vee (\bigvee_{j=1}^{\delta}(\tau_j \wedge \bigwedge_{i=j+1}^{\delta} \rho_i))$, where, for $k \in [1, \delta]$, ρ_k and τ_k are obtained from ρ and τ, respectively, by replacing every position formula φ by $\mathrm{Y}^k\varphi$, every $@b$ by $@a = Y^k@b$ and every $\overline{@b}$ by $\neg @a = Y^k@b$. It can be observed that this formula has the intended meaning (that is, the semantics obtained by using $-\delta$ in the above semantics definition). $\rho \mathrm{S}_{@a}\tau$ is defined analogously.

First of all, we will see that the above mentioned restriction for class formulas $\rho \mathrm{U}^{=}\tau$ is indeed crucial. More precisely, if we allow positive attribute tests in the target formula of a formula $\rho\, \mathrm{U}_{@a}\, \tau$ then the logic becomes undecidable even if the intermediate formulas are restricted to position formulas.

▶ **Theorem 6.** *Let $\mathcal{L}$ denote the extension of BD-LTL by the formation rule $\varphi ::= \chi\, \mathrm{U}_{@a}\, \chi$, where χ denotes* U*-subformulas such that*

- *all intermediate formulas are position formulas and*
- *all target formulas are of the form $@a \wedge \varphi$ with a position formula φ.*

Then, satisfiability of $\mathcal{L}$ on finite (or infinite) attributed words is undecidable. This holds even for 1-attributed words.

The proof is again by a reduction from the non-emptiness problem for Minsky two counter automata [21]. As Theorem 6 does not leave much room for extensions of $\mathrm{U}_{@a}$ operators with positive attribute tests in the target formula we focus on negative attribute tests in target formulas. However, as $\rho \mathrm{U}^{\delta}_{@a}(\tau_1 \vee \tau_2) \equiv (\rho \mathrm{U}^{\delta}_{@a}\tau_1) \vee (\rho\, \mathrm{U}_{@a}\, \tau_2)$ and position formulas are closed under conjunctions it is clearly sufficient to consider target formulas of the form $\varphi \wedge \overline{@b_1} \wedge \cdots \wedge \overline{@b_k}$. Unfortunately, at this point our techniques can only deal with the case $k = 1$.

We turn our attention now to the intermediate formulas ρ. We recall that in the case of positive attribute tests in target formulas even position formulas as intermediate formulas yield undecidability. In the case of (single) negative attribute tests in target formulas we can allow arbitrary intermediate position formulas.

Furthermore, we can add positive and negative attribute tests, but only in a limited way. More precisely, we define the logic XD-LTL by adding $\varphi ::= \chi \mathrm{U}^{\delta}_{@a}\chi' \mid \chi \mathrm{S}^{\delta}_{@a}\chi'$, to the formation rules of BD-LTL and requiring that

1. χ is restricted to formulas of the form $\rho \vee (@b \wedge \rho^{=}) \vee (\overline{@b} \wedge \rho^{\neq})$ where $\rho^{=}, \rho^{\neq}$ are position formulas and $\rho^{\neq}$ logically implies[5] $\rho^{=}$, and

[5] Readers who prefer a syntactical criterion might think of a formula $\rho^{=}$ of the form $\varphi \vee \rho^{\neq}$.

2. χ' is restricted to formulas of the form $\overline{@b} \wedge \tau$, where τ is a position formula.

Intuitively, $\rho^{=}$ constrains positions where $@b$ equals the current value of $@a$ whereas $\rho^{\neq}$ constrains those where it does not. The requirement that $\rho^{\neq}$ implies $\rho^{=}$ is needed for the proof of Theorem 8.

Clearly XD-LTL strictly extends BD-LTL and is contained in $\mathrm{LTL}_1^{\downarrow}$. Further it strictly extends two-variable logic on 1-attributed words.

Following the general idea of the decidability proof for BD-LTL we first show decidability of satisfiability for 1-attributed words and reduce the general case to this one.

▶ **Theorem 7.** *Satisfiability for XD-LTL on finite 1-attributed words is decidable.*

Proof. (Sketch.) The proof basically extends the proof of Theorem 2 for formulas of type $\psi = (@a \wedge \rho^{=}) \vee (\overline{@a} \wedge \rho^{\neq}) \mathrm{U}^{\delta}_{@a} (\overline{@a} \wedge \tau)$. Note that in the case of 1-attributed words, any additional disjunct ρ in the intermediate formula can be pushed into the disjunction by or-ing it with both $\rho^{=}$ and $\rho^{\neq}$.

For a given position i with data value d fulfilling $w, i \models \psi$ we call the minimal position j that fulfills $\rho^{\neq}$ and has a data value different from d, the *ψ-shepherd for i*. We write $H(j)$ for the herd of j, that is the set of positions for which j is a ψ-shepherd. With each τ-position j we associate a set $S(j)$ of *special positions*. Roughly speaking, if i is in the herd of j, then positions in $[i, j)$ with the same data value as i are special. The special interval $I(j)$ for a shepherd j is the minimal interval containing $S(j)$. Two crucial observations are that (1) all positions in $S(j)$ have the same data value and (2) $|I(j) \cap I(j')| \leq \delta$ for $j \neq j'$.

In a nutshell, the idea for the construction of the data automaton for ψ is as follows. Besides the propositions for the subformulas, we use further propositions of the form H, e^{+} and e^{-} with the intention that for each shepherd marked by τ, the end points of the special interval are marked by e^{+} and e^{-}, respectively, and all positions in $H(j)$ are marked by H.

As we are testing satisfiability, we can safely assume that all those propositions are already present in the input word, but their consistency has to be verified by the automaton. The automaton then checks that for each τ-position j the corresponding e^{+}- and e^{-}-positions are as intended. Further it guesses and checks all other positions in $S(j)$. Finally consistency of H- and τ-positions is verified.

As for BD-LTL, special attention is needed for $\delta \neq 0$. For the detailed proof, we refer the reader to the full version of the paper [17].

◀

By a straightforward extension of the proof of Theorem 1 we get the following.

▶ **Theorem 8.** *Satisfiability for XD-LTL on finite attributed words is decidable.*

6 Conclusion

We conclude by stating some questions that should be investigated further. We would be interested to understand the exact border of undecidability. At this point, it is not exactly clear which kinds of intermediate and target formulas can be allowed for $\mathrm{U}^{\delta}_{@a}$. It would also be interesting to compare our logics with other logics that can deal with values, particularly with guarded LTL-FO of [14]. Further investigations could try to identify fragments with more reasonable complexity and try to add more arithmetics to the data domain.

References

1 R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.

2 T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. D. Zuck. Parameterized verification with automatically computed inductive assertions. In *CAV*, volume 2620 of *Lecture Notes in Computer Science*, pages 221–234, 2001.

3 H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5):702–715, 2010.

4 M. Bojanczyk. Personal communication, 2006.

5 M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS*, pages 7–16. IEEE Computer Society, 2006.

6 A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *Fundamentals of Computation Theory*, volume 4639 of *Lecture Notes in Computer Science*, pages 1–22. Springer Berlin / Heidelberg, 2007.

7 P. Bouyer. A logical characterization of data languages. *Inf. Process. Lett.*, 84(2):75–85, 2002.

8 P. Bouyer, A. Petit, and D. Therien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003.

9 S. Demri. LTL over integer periodicity constraints. In *FoSSaCS*, pages 121–135, 2004.

10 S. Demri, D. D'Souza, and R. Gascon. A decidable temporal logic of repeating values. In S. N. Artëmov and A. Nerode, editors, *LFCS*, volume 4514 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 2007.

11 S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. In *LICS '06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 17–26, Washington, DC, USA, 2006. IEEE Computer Society.

12 S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

13 S. Demri, R. Lazić, and D. Nowak. On the freeze quantifier in constraint LTL: Decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.

14 A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.

15 D. Figueira and L. Segoufin. Future-looking logics on data words and trees. In R. Královic and D. Niwinski, editors, *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 331–343. Springer, 2009.

16 M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

17 A. Kara, T. Schwentick, and T. Zeume. Temporal logics on words with multiple data values. Available from arXiv:1010.1139, 2010.

18 F. Laroussinie and P. Schnoebelen. A hierarchy of temporal logics with past. *Theor. Comput. Sci.*, 148(2):303–324, 1995.

19 A. Lisitsa and I. Potapov. Temporal logic with predicate lambda-abstraction. In *TIME 2005*, pages 147–155, 2005.

20 A. Manuel. Two orders and two variables. In *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 513–524, 2010.

21 M. L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

22 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

23 T. Schwentick and T. Zeume. Two-variable logic with two order relations. In *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 499–513, 2010.

24 T. Tan. On pebble automata for data languages with decidable emptiness problem. In *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 712–723, 2009.