PDL on infinite alphabet

Mathieu Lehaut, supervised by Benedikt Bollig and Paul Gastin

LSV, ENS Cachan

The general context

In the context of verification, one is often interested in finding decidable logics to be able to express and then check specifications. Here we study logics on datawords, i.e. words on an infinite alphabet, which can be used to model executions of a program manipulating data. Furthermore, the alphabet comes with a total order. Several logics have been studied in this context, but they often lack the ability to use the total order. In the meanwhile, new kinds of automata have been created in order to work on datawords, such as register automata.

The research problem

We study a new logic which is an extension of PDL on words with a constructor that allows to compare the data of two positions with respect to the total order. We are interested in the decidability of the satisfiability problem, which is: given a formula ϕ in the logic, is there a dataword satisfying ϕ? In the context of verification, this allows to check that a specification does not contain contradictions.

Your contribution

We compare our new logic with the first order logic with two variables that has already been studied. From this comparison, we deduce that our logic is undecidable. We then study a fragment of our logic on which we can get back decidability by reducing to the emptiness problem for some kind of register automata, which has been proven decidable.

Arguments supporting its validity

Although the complete logic is undecidable, the decidable fragment we study is still very expressive, and, contrary to other logics, allows the use of the total order on data. We give some example properties/specifications which can be expressed in this fragment.

Summary and future work

Our works brings a new decidable logic which can be used to express new kinds of specifications in the context of datawords. The next step is to extend this to infinite datawords and datatrees to increase the number of behaviours that can be modeled and then specified with the logic.
Note: this report is in english as it may serve as the basis for a future article.

1 Introduction

Context In the context of verification, one needs a model and a logic to write specifications on the model. Here, we are interested in the model of words on an infinite alphabet consisting of a finite part and an infinite part which comes equipped with a total order. A dataword is a word on such an alphabet, that is, each position of the word contains a couple of letters, one from the finite part and the other from the infinite part.
This model has been studied since [9] in which a new model of automata, able to deal with infinite alphabets, has been introduced. This model is interesting in that it allows to model the execution of a program manipulating data: the successive positions correspond to the successive states of the program, where a state consists of a control state (= letter from the finite part) and the data being manipulated (= letter from the infinite part). One can also see a dataword as a snapshot of an execution of a distributed algorithm, where each position in the word corresponds to a single process, with the corresponding control state and data.

Automata Several kinds of automata for datawords have been studied following [9]: register automata and pebble automata in [11] (as well as some variations on them), or strong automata and progressive grid automata in [3]. A register automaton is an automaton which can store the data of the current position in one of its registers (of which there is a fixed number), and later in the word use the saved value to make comparisons with the new current data, then possibly store a new value in the register, and so on. In our works, we will focus on alternating one-register automata, that is, register automata with only one register to store data values, but that can use alternation (as in alternating automata for words).

Logics With this model of datawords also come new logics to express properties or write specifications. One of the minimal requirements for these new logics is that the satisfiability problem must be decidable: given a formula ϕ, is there a dataword satisfying ϕ? This problem makes sense in the field of verification as it asks whether a specification is actually attainable, i.e. that it does not contradict itself. Among the decidable logics, perhaps one of the most studied is the first order logic with two variables (FO^2) explored in [2]. Its decidability depends on which predicates are allowed: one can allow a linear order and successor relation on positions and an equality predicate on data [2], at the cost of not being able to use the total order on data (adding it makes the logic undecidable), or allow only a linear order on positions and the total order on data as in [12], at the cost of not being able to link two consecutive positions. Other decidable logics include LTL with the freeze operator [4] and PathLogic [5]. Both of these also cannot refer to the total order on data.

Our contribution In this report we explore a new logic which we call DataPDL which is an extension of PDL, first defined in [7], adapted to the setting of datawords by adding an operator that allows to compare the data at two positions

according to the total order on data (or equality, inequality, etc.). This logic is two-way, that is, it can navigate in the word in both directions. It is inspired from the logic DataPDL defined in [1] which works on tables (2D) as opposed to words (1D) here.

Outline of this report In Section 2, we write down all relevant definitions. In Section 3 we first compare DataPDL to FO^2 and conclude that DataPDL is undecidable. We then propose some restrictions to get a fragment $\mathrm{DataPDL}^{\ominus}$. In Section 4 we prove that this fragment is decidable by reduction to alternating one-register automata. In Section 5 we discuss one of the restriction defining $\mathrm{DataPDL}^{\ominus}$.

2 Definitions

2.1 Dataword

Let Σ be a finite alphabet of *letters* and $\mathcal{D}$ an infinite domain of *data values* equipped with a total order $<$. We note *Comp* the set $\{\leq, <, =, \neq, >, \geq\}$ of comparison operators.
A (finite) dataword is a non-empty word $w = w_0 \cdots w_{n-1}$ on $(\Sigma \times \mathcal{D})^*$. We denote by $w[i]$ the i-th letter of w, i.e. the first component of w_i, and by $w.d[i]$ the i-th data value of w, i.e. the second component of w_i.

2.2 FO^2

We consider the first-order logic on datawords, whose atomic predicates are: $x = y + 1$, $x < y$, $d(x) < d(y)$, and $a(x)$ for all $a \in \Sigma$. The variables $(x, y, \dots)$ refer to positions in the word. Formula $x = y + 1$ means that x is the position immediately following y, and $x < y$ means that y is a position following x, though not necessarily the next one. Similarly, $d(x) < d(y)$ means that the data value at position x is smaller that the data value at position y. Finally, $a(x)$ means that the letter at position x is an a.
These atomic predicates can then be combined with the standard negation ($\neg$), disjunction ($\vee$) and existential quantification ($\exists x.\phi$). From these we can easily define conjunction ($\wedge$), implication ($\Rightarrow$), and universal quantification ($\forall x.\phi$) in the usual way, as well as atomic predicates for all comparison operators in *Comp*: $d(x) \leq d(y)$ is $\neg(d(y) < d(x))$, $d(x) = d(y)$ is $d(x) \leq d(y) \wedge d(x) \geq d(y)$, and so on.
The full first-order logic is too expressive to be used, so we focus on the fragment noted FO^2 where formulas can only use two variables names. For instance, here are some properties that can be expressed in FO^2:

$$\forall x. \forall y. \, (x < y \Rightarrow d(x) < d(y))$$

means that the data values in the word are ordered, while

$$\exists x. \mathit{leader}(x) \wedge \forall y. \, (x \neq y \implies (\neg \mathit{leader}(y) \wedge d(y) < d(x)))$$

express that there is a unique position labelled *leader* which holds the maximal data value.

2.3 DataPDL

The logic DataPDL on Σ and $\mathcal{D}$ is given by the following grammar:

$$\phi ::= a \mid \neg\phi \mid \phi \vee \phi' \mid \langle\pi\rangle\phi \mid \langle\pi\rangle \bowtie \langle\pi'\rangle$$

$$\pi ::= \{\phi\}? \mid \leftarrow \mid \rightarrow \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*$$

with $a \in \Sigma$ and $\bowtie \in \mathit{Comp}$.
We call ϕ a *local formula* and π a *path formula* (or simply a *path*). Intuitively, a local formula will be tested at some position in the word, and a path formula links two positions.
A local formula $\phi = a$ is true if the letter at the current position is a. $\neg$ and $\vee$ are the standard negation and disjunction operators. $\langle\pi\rangle\phi$ means "from the current position, there is a position we can reach following path π, such that the formula ϕ holds at this position". Similarly, a formula $\langle\pi\rangle = \langle\pi'\rangle$ means "from the current position, there exist two positions, one that can be reached following π, and the other following π', such that the data at these two positions are equal."
A path formula is built with $\leftarrow$ (resp. $\rightarrow$) which refers to the previous (resp. next) position in the word, $+, \cdot, *$ which are the standard non-deterministic choice, composition and star operators, and guards $\{\phi\}?$ which allow testing local formula in the middle of the path.
We now give the formal semantics. We note $w, i \models \phi$ to express that the word w at position i satisfies ϕ. Given a word w, the semantics of a path formula π, noted $[\![\pi]\!]_w$, is a binary relation between positions. Let us define these two notions inductively:

- $w, i \models a$ if $w.[i] = a$
- $w, i \models \neg\phi$ if $w, i \not\models \phi$
- $w, i \models \phi \vee \phi'$ if $w, i \models \phi$ or $w, i \models \phi'$
- $w, i \models [\pi]\phi$ if for all j such that $(i, j) \in [\![\pi]\!]_w$, $w, j \models \phi$
- $w, i \models \langle\pi\rangle \bowtie \langle\pi'\rangle$ if there exists j and j' such that $(i, j) \in [\![\pi]\!]_w$, $(i, j') \in [\![\pi']\!]_w$, and $w.d[j] \bowtie w.d[j']$

- $[\![\{\phi\}?]\!]_w = \{(i, i) \mid w, i \models \phi\}$
- $[\![\leftarrow]\!]_w = \{(i, i-1) \mid 1 \leq i < |w|\}$
- $[\![\rightarrow]\!]_w = \{(i, i+1) \mid 0 \leq i < |w| - 1\}$
- $[\![\pi + \pi']\!]_w = [\![\pi]\!]_w \cup [\![\pi']\!]_w$
- $[\![\pi \cdot \pi']\!]_w = [\![\pi']\!]_w \circ [\![\pi]\!]_w$
- $[\![\pi^*]\!]_w = \bigcup_{n \geq 0} ([\![\pi]\!]_w)^n$, where $([\![\pi]\!]_w)^n = [\![\pi]\!]_w \circ \cdots \circ [\![\pi]\!]_w$ n times

A dataword w satisfies ϕ, noted $w \models \phi$, if $w, 0 \models \phi$.
We define the usual macros: $\top = a \vee \neg a$, $\bot = \neg\top$, $[\pi]\phi = \neg\langle\pi\rangle\neg\phi$, $\phi \wedge \phi' =$

$\neg(\neg\phi \vee \neg\phi')$ for local formulas, $\pi^+ = \pi \cdot \pi^*$, and $\varepsilon = \{\top\}?$ for the path formula which does not move.
We give two example formulas corresponding to the two examples given in the FO^2 section:

$$[\rightarrow^*]\neg\left(\langle\varepsilon\rangle > \langle\rightarrow^+\rangle\right)$$

for the "ordered data values" property, and

$$\begin{aligned}&\langle(\{\neg leader\}? \cdot \rightarrow)^*\rangle\ (leader \wedge [\rightarrow^+]\ \neg leader)\\ \wedge\ &\neg\langle\rightarrow^* \cdot \{leader\}?\rangle \leq \langle\rightarrow^* \cdot \{\neg leader\}?\rangle\end{aligned}$$

for the "unique and maximal leader" property.

2.4 Alternating one-register automaton

For any set S, we note $\mathcal{B}^+(S)$ the set of positive boolean combinations of S, built as follows:

$$\phi ::= \top \mid \bot \mid s \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$$

for all $s \in S$ and $\phi_1, \phi_2 \in \mathcal{B}^+(S)$.
We define when a multiset T of elements of S satisfies $\phi \in \mathcal{B}^+(S)$, written $T \models \phi$:

- $\emptyset \models \top$
- $\{s\} \models s$
- $T \models \phi_1 \vee \phi_2$ if $T \models \phi_1$ or $T \models \phi_2$
- $T \models \phi_1 \wedge \phi_2$ if $T = T_1 \uplus T_2$ and $T_1 \models \phi_1$ and $T_2 \models \phi_2$

We note $Comp^? = Comp \cup \{nop\}$ and $Store^? = \{\downarrow, nop\}$.

An *alternating one-register automaton* (A1RA) on Σ and $\mathcal{D}$ is a tuple $\mathcal{A} = (Q, q_0, F, \Delta)$ with Q a set of states, q_0 an initial state, F a set of final states, and $\Delta \subseteq Q \times \Sigma \times Comp^? \times \mathcal{B}^+(Q \times Store^?)$ is a finite transition relation.
Intuitively, an A1RA acts as an alternating automaton on words which can also store one value from $\mathcal{D}$ in its register, and use it to compare with the data that appear in the dataword it is reading. A transition $\delta = (q, a, \bowtie, \phi)$ can be taken if the current state is q, the current letter is a, and if $\bowtie \in Comp$ then the current data value is compared to the value stored in the register (if $\bowtie = nop$, we skip this step); if these hold then the automaton continues the computation in the next position in a set of states satisfying ϕ, where each branch $(q, \downarrow)$ stored the value from the previous position into the register, and each branch (q, nop) kept the register value unchanged.
Its formal semantics is as follows.

Definition 1. *A* run ρ *on a dataword* w *is a* $(Q \times \Delta \times \mathcal{D})$*-labelled tree, of height at most the size of* w*, such that every node* $(q, \delta = (q, a, \bowtie, \phi), d)$ *of height* $i < |w|$ *verifies:*

- $a = w.[i]$

– *if* $\bowtie \in Comp$*, then* $w.d[i] \bowtie d$ *holds.*
– *there is a multiset* T *of couples from* $Q \times Store^?$ *satisfying* ϕ *and a bijection* α *from* T *to the children of this node such that* $\alpha(q', ins)$ *is labelled* (q', δ', d') *for some* $\delta' \in \Delta$*, with* $d' = w.d[i]$ *if* $ins = \downarrow$ *and* $d' = d$ *otherwise.*

Also, the root must be labelled $(q_0, \delta, w.d[0])$ *for some* $\delta \in \Delta$*.*

A run is said to be *accepting* if all its leaves of height $|w|$ are in $F \times \Delta \times \mathcal{D}$. As usual, a dataword w is *accepted* by $\mathcal{A}$ if there exists an accepting run on w, and the *language* of $\mathcal{A}$ is the set of datawords accepted by $\mathcal{A}$.

Proposition 1. *The emptiness problem for A1RA is decidable.*

Proof. See [6]. The proof links register automata on datawords with clock automata on times words. The emptiness problem for alternating one-clock timed automata has been proven decidable in [10]. □

Also, as they are alternating, it is easy to check that A1RA are closed under boolean operations.

Proposition 2. *A1RA are closed under union, intersection and complement.*

Proof. Union and intersection are done by disjunction and conjunction, while complement is obtained by exchanging final states with non-final states, $\top$ with $\bot$, and $\vee$ with $\wedge$ in all transitions.

3 Properties of DataPDL

3.1 Relation to FO²

We compare the expressive power of DataPDL with FO^2. It is easy to see that there are properties that can be expressed in DataPDL but not in FO^2. For instance, "there are two positions with the same data value, and between these positions there are no b" corresponds to the DataPDL formula $\langle \rightarrow^* \rangle (\langle \varepsilon \rangle = \langle \rightarrow \cdot (\{\neg b\}? \cdot \rightarrow)^* \rangle)$, but cannot be expressed in FO^2 (the intuition is that you have to use the two variables to remember both positions, and so you would need a third one to go between those). Even without using data values, one can express the property "the word has even length" in DataPDL but not in FO^2. Conversely, we study the other direction.

Proposition 3. *For any* FO^2 *formula, one can effectively build a DataPDL formula which is equivalent with respect to satisfiability.*

Proof. Let $\phi \in \mathrm{FO}^2$ on alphabet Σ. We first transform ϕ into an equivalent formula ϕ' in *Scott normal form*. See [8] for more details. ϕ' is a FO^2 formula on an extended alphabet Σ':

$$\phi' = \forall x \forall y . \chi_0 \wedge \bigwedge_{1 \leq i \leq n} \forall x \exists y . \chi_i$$

where χ_0 and all χ_i are quantifier-free FO^2 formula. Now we only have two kinds of formulas to translate: the $\forall x \forall y.\chi$ one, and the $\forall x \exists y.\chi$ one. Without loss of generality, we can ask that χ is in *Disjunctive normal form*, i.e. χ is a disjunction of clauses, and a clause is a conjunction of atomic formulas or their negations. There are four kinds of atomic formulas:

1. Constraints on the letter of x $(a(x), \neg b(x), \dots)$
2. Constraints on the letter of y (same with y)
3. Constraints on the relative position of x and y $(x < y, x = y + 1, \dots)$
4. Constraints on the relative data values of x and y $(d(x) < d(y), d(x) \neq d(y), \dots)$

A clause cl is a conjunction of these four kinds of constraints.
Let the constraint cube be the *finite* set $\mathcal{C} = \Sigma' \times \Sigma' \times \{\ll, -1, =, +1, \gg\} \times \{<, =, >\}$. A clause cl translates to a subset $\mathcal{C}_{cl}$ of points of $\mathcal{C}$ which reflects its constraints: for instance, with $\Sigma' = \{a, b, c\}$, $cl = \neg a(y) \wedge \neg b(y) \wedge y < x \wedge d(x) \leq d(y)$ translates to $\mathcal{C}_{cl} = \{a, b, c\} \times \{c\} \times \{+1, \gg\} \times \{<, =\}$.
Given some dataword w and two positions x and y of w, x and y satisfy the constraints of a unique point c in $\mathcal{C}$. If moreover, c is in $\mathcal{C}_{cl}$, then we know that cl is satisfied by these x and y (and the converse holds). We define $\mathcal{C}_{good}$ the union over all clauses cl of χ of such subsets $\mathcal{C}_{cl}$. To each point $c = (a, b, p, \bowtie) \in \mathcal{C}$ we associate a DataPDL comparison test:

$$\phi_c = \langle \{a\}? \rangle \bowtie \langle \pi_p \cdot \{b\}? \rangle$$

where π_p depends on p:

- $\pi_{\ll} = \leftarrow \cdot \leftarrow^+$
- $\pi_{-1} = \leftarrow$
- $\pi_{=} = \varepsilon$
- $\pi_{+1} = \rightarrow$
- $\pi_{\gg} = \rightarrow \cdot \rightarrow^+$

The meaning of ϕ_c is that, for a dataword w, ϕ_c is satisfied at position x if and only if there exists a position y such that x and y satisfy the constraints of c.
And so, we translate formulas $\forall x \exists y \chi$ as

$$\phi_{\forall\exists} = [(\leftarrow + \rightarrow)^*] \bigvee_{c \in \mathcal{C}_{good}} \phi_c$$

and the formula $\forall x \forall y \chi$ (seen as $\forall x \neg \exists y \neg \chi$) as

$$\phi_{\forall\forall} = [(\leftarrow + \rightarrow)^*] \neg \left(\bigvee_{c \notin \mathcal{C}_{good}} \phi_c \right)$$

The $[(\leftarrow + \rightarrow)^*]$ part in both formulas obviously corresponds to the $\forall x$. We claim that $\forall x \exists y \chi$ and $\forall x \forall y \chi$ accept the same datawords as $\phi_{\forall\exists}$ and $\phi_{\forall\forall}$ respectively.

$$\begin{aligned}
w \models \phi_{\forall\exists} &\iff \forall i, \exists c \in \mathcal{C}_{good}, w, i \models \phi_c \\
&\iff \forall i, \exists c \in \mathcal{C}_{good}, \exists j, i, j \models c \\
&\iff \forall i, \exists j, \exists c \in \mathcal{C}_{good}, i, j \models c \\
&\iff \forall i, \exists j, \exists cl, i, j \models cl \\
&\iff \forall i, \exists j, i, j \models \chi
\end{aligned}$$

$$\begin{aligned}
w \models \phi_{\forall\forall} &\iff \forall i, \neg\exists c \notin \mathcal{C}_{good}, w, i \models \phi_c \\
&\iff \forall i, \neg\exists c \notin \mathcal{C}_{good}, \exists j, i, j \models c \\
&\iff \forall i, \neg\exists j, \exists c \notin \mathcal{C}_{good}, i, j \models c \\
&\iff \forall i, \neg\exists j, \neg\exists c \in \mathcal{C}_{good}, i, j \models c \\
&\iff \forall i, \neg\exists j, \neg\exists cl, i, j \models cl \\
&\iff \forall i, \neg\exists j, \neg\chi \iff \forall i, \forall j, \chi
\end{aligned}$$

□

3.2 Undecidability

The emptiness (satisfiability) problem for FO^2 has been shown undecidable in [2]. Given Proposition 3, the following result is not surprising:

Proposition 4. *The emptiness problem for DataPDL is undecidable.*

We still give a proof, adapted from the one for FO^2, to illustrate what kind of DataPDL formula is enough to get undecidability.

Proof. We reduce Post's Correspondence Problem. Let $\{(u_i, v_i) \mid 1 \leq i \leq k\}$ be an instance of PCP on an alphabet Σ. Let $\bar{\Sigma}$ be a copy of Σ. For a word $w \in \Sigma^*$, we define $\bar{w} \in \bar{\Sigma}^*$ as the word w where each letter is replaced by its copy from $\bar{\Sigma}$.
A solution $(i_1, ..., i_n)$ of PCP will be encoded as a dataword:

- whose projection on $\Sigma \cup \bar{\Sigma}$ is $u_{i_1} \bar{v_{i_1}} ... u_{i_n} \bar{v_{i_n}}$
- such that the data on the Σ-subword are pairwise different, in increasing order, and similarly for the $\bar{\Sigma}$-subword, and with both sequences of data equal.

For example, if the instance is $\{(ab, baa), (a, ba), (aba, a)\}$, the solution $3 \cdot 1 \cdot 2$ will be encoded as the dataword:

$$(a,1)(b,2)(a,3)(\bar{a},1)(a,4)(b,5)(\bar{b},2)(\bar{a},3)(\bar{a},4)(a,6)(\bar{b},5)(\bar{a},6)$$

Now we build a formula ϕ such that ϕ accepts exactly these encodings, meaning that the language of ϕ is empty if and only if this instance of PCP has no

solution.
We use Σ as an abbreviation for $\bigvee_{a\in\Sigma} a$, and same for $\bar{\Sigma}$. Without loss of generality, we can assume that in the solution, each letter from u_i happens before its corresponding letter from v_j.
ϕ has a few points to check:

1. The string projection of w is in $\{u_i\bar{v_i} \mid 1 \leq i \leq k\}^+$:
$$[\rightarrow^* \{\Sigma \wedge [\leftarrow]\bar{\Sigma}\}?] \bigvee_{i\in[1,k]} (u_i\bar{v_i})$$
2. Data on Σ are unique:
$$[\rightarrow^* \Sigma?]\neg(\langle\varepsilon\rangle = \langle\rightarrow^* \Sigma?\rangle)$$
3. Data on $\bar{\Sigma}$ are unique (similar as the previous one)
4. Each letter on Σ has a corresponding letter in $\bar{\Sigma}$:
$$[\rightarrow^* \Sigma?] \bigvee_{a\in\Sigma} (a \wedge \langle\varepsilon\rangle = \langle\rightarrow^* \bar{a}?\rangle)$$
5. Same for $\bar{\Sigma}$ (similar, with the last $\rightarrow^*$ replaced by $\leftarrow^*$)
6. Data on Σ are on increasing order:
$$[\rightarrow^* \Sigma?]\neg(\langle\varepsilon\rangle > \langle\rightarrow^* \Sigma?\rangle)$$
7. Same for $\bar{\Sigma}$ (again, similar)

ϕ is the conjunction of all these formulas. We can easily check that a word is accepted by ϕ if and only if it is a valid encoding of a PCP solution. □

3.3 Restrictions

Knowing that FO^2 is undecidable, some restrictions have been explored to regain decidability. One can forbid using the order on data values, and instead only allow equality (and inequality) tests, as in [2]. One can also forbid the successor relation on positions, leaving only the orders on positions and values, as explored in [12].
For our logic DataPDL, we propose some restrictions that are orthogonal to that approach. They will restrict the power of path formulas in comparison tests:
Consider the fragment of DataPDL where comparison test $\langle\pi\rangle \bowtie \langle\pi'\rangle$ follow two restrictions:

- π and π' both use only one direction, left or right (not necessarily the same for both paths). Which means that if $\leftarrow$ appears in π, then there are no $\rightarrow$ in π, except possibly within guards $\{\phi\}?$.
- If π (resp. π') is left-oriented, then π (resp. π') is *non-ambiguous*. A path π is non-ambiguous if for all datawords w, for all positions x of w, $|\{y \mid (x,y) \in [\![\pi]\!]_w\}| \leq 1$. In other words, following the path cannot lead to two different positions. Right-oriented paths are not restricted.

These restrictions can be enforced syntaxically by using the following grammar:

$$\begin{aligned}
\phi &::= a \mid \neg\phi \mid \phi \vee \phi' \mid \langle\pi\rangle\phi \mid \langle\pi^d\rangle \bowtie \langle\pi^{d'}\rangle \\
\pi &::= \{\phi\}? \mid \leftarrow \mid \rightarrow \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^* \\
\pi^{\leftarrow} &::= \{\phi\}? \mid \leftarrow \mid \pi^{\leftarrow} \cdot \pi^{\leftarrow} \mid F_{\phi}^{\pi^{\leftarrow}} \\
\pi^{\rightarrow} &::= \{\phi\}? \mid \rightarrow \mid \pi^{\rightarrow} + \pi^{\rightarrow} \mid \pi^{\rightarrow} \cdot \pi^{\rightarrow} \mid (\pi^{\rightarrow})^*
\end{aligned}$$

where $d, d' \in \{\leftarrow, \rightarrow\}$, and $F_{\phi}^{\pi^{\leftarrow}}$ is a macro that stands for $(\{\neg\phi\}? \cdot \pi^{\leftarrow})^* \cdot \{\phi\}?$, i.e. "the first position satisfying ϕ using $\pi^{\leftarrow}$ as steps" (which is obviously non-ambiguous).

Let DataPDL$^{\ominus}$ be this fragment. The two example formulas given earlier belong in this fragment. Our main theorem, which we prove in the next section, is the following.

Theorem 1. *The emptiness problem for DataPDL$^{\ominus}$ formulas is decidable.*

4 Decidability of DataPDL$^{\ominus}$

4.1 The construction for a restricted case

Let ϕ be a DataPDL$^{\ominus}$ formula. We want to construct an A1RA $\mathcal{A}_\phi$ such that the language $\mathcal{L}(\mathcal{A}_\phi)$ of $\mathcal{A}_\phi$ is empty if and only if the language $\mathcal{L}(\phi)$ of ϕ is empty.
If ϕ has no data comparison tests, then it is simply a property of words. It is known that for words, two-way automata are as expressive as one-way automata. Therefore, in this case, we simply need to translate the PDL formula into a two-way automaton, turn this into a one-way automaton, and say this automaton is actually an A1RA which does not use its register, and we win.
Now we explain the intuition on how to deal with comparison tests. To simplify the proof, we add two new restrictions on comparison tests $\langle\pi\rangle \bowtie \langle\pi'\rangle$:

- The left hand side path π must be ε
- π' must not contain nested comparison tests: π' cannot contain guards which themselves contain a comparison test subformula.

These additional restrictions will be removed later.

ϕ is a fixed formula. It has a finite number n of subformulas which are comparison tests. Let $\phi_0, ..., \phi_{n-1}$ be those subformulas. The idea is to isolate these comparison tests by replacing them with new letters that will mark at which positions the comparison test holds. Let $C = \{c_0, ..., c_{n-1}\}$ be n new letters, with each letter c_i associated to the subformula ϕ_i. For instance, let

$$\phi = [\rightarrow^* \cdot \{\neg b\}?]\,(\langle\varepsilon\rangle = \langle\rightarrow\rangle \wedge \langle\leftarrow^*\rangle a)$$

on alphabet $\Sigma = \{a, b\}$. Here, ϕ_0 is the subformula $\langle\varepsilon\rangle = \langle\rightarrow\rangle$.
We will extract the comparison test by adding a new letter c_0 in that way:

$$\phi' = ([\rightarrow^* \cdot \{\neg b\}?]\,(c_0 \wedge \langle\leftarrow^*\rangle a)) \wedge ([\rightarrow^*]\,(c_0 \Leftrightarrow \langle\varepsilon\rangle = \langle\rightarrow\rangle))$$

on alphabet $\Sigma' = \{a, b\} \times 2^{\{c_0\}}$. ϕ and ϕ' are equivalent with respect to satisfiability, however ϕ' has a nice structure: the left part is simply a formula involving no data, so the reasoning above applies, and we only have a specific kind of comparison tests in the right part, which are of the form $[\rightarrow^*](c_i \Leftrightarrow \langle\varepsilon\rangle \bowtie \langle\pi\rangle)$. In other words, we need to be able to check that the markings are correct.
Formally, ϕ being fixed, let $\Sigma_\phi = \Sigma \times 2^C$, and $\tau : (\Sigma \times \mathcal{D})^* \rightarrow (\Sigma_\phi \times \mathcal{D})^*$ the transformation such that

- $\tau(w).d[i] = w.d[i]$ (data are unchanged)
- $\tau(w).[i] = (w.[i], \{c_j \mid c_j \text{ holds at position } i\})$

Now let ϕ_τ be the formula that is ϕ in which each comparison subformula c_j is replaced by the letter test c_j (for example, if $\phi = [\rightarrow^* \cdot\{a\}?]\langle\varepsilon\rangle \neq \langle\leftarrow\rangle$, then $\phi_\tau = [\rightarrow^* \cdot\{a\}?]c_0$). Then we define

$$\phi' = \phi_\tau \wedge \bigwedge_{0 \leq i < n} [\rightarrow^*](c_i \Leftrightarrow \phi_i)$$

Proposition 5. $\mathcal{L}(\phi') = \tau(\mathcal{L}(\phi))$

Proof. $\supseteq$: by definition of τ
$\subseteq$: Let $w' \in \mathcal{L}(\phi')$, and w be its projection on $(\Sigma \times \mathcal{D})^*$. As w' satisfies the right part of ϕ', the markings of all c_i are correct, thus $w' = \tau(w)$. We show that w satisfies ϕ recursively on the structure of ϕ; as ϕ and ϕ_τ have the same structure, only the base case is interesting. So we have that w' satifies some letter test c_i at some position, and we want to show that w satisfies the associated comparison test ϕ_i at the same position. But we also know that w' satisfies formula $[\rightarrow^*](c_i \Leftrightarrow \phi_i)$ and that the data of w and w' are the same, which imply that w satisfies ϕ_i at that position. □

Corollary 1. *$\mathcal{L}(\phi)$ is empty if and only if $\mathcal{L}(\phi')$ is empty.*

We describe how to build an A1RA whose language is the same as ϕ'. The left part, ϕ_τ, is a formula without data. Following the remark given in the beginning of this section, there is an A1RA $\mathcal{A}_\tau$ whose language is $\mathcal{L}(\phi_\tau)$. What remains is, for each $0 \leq i < n$, to build an A1RA $\mathcal{A}_i$ whose language is the language of subformula $[\rightarrow^*](c_i \Leftrightarrow \phi_i)$, or said otherwise, $\mathcal{A}_i$ checks that the markings for c_i are correct. The intersection of $\mathcal{A}_\tau$ and all $\mathcal{A}_i$ then gives the language $\mathcal{L}(\phi')$. Let us describe how those $\mathcal{A}_i$ are built.

As A1RA are inherently one-way, the way we deal with left-oriented and right-oriented paths will be different. Let us now describe how to build $\mathcal{A}_i$ in the two different cases.
Let $\phi_i = \langle\varepsilon\rangle \bowtie \langle\pi^{\leftarrow}\rangle$, and c_i its corresponding marking. π is a left-oriented PDL path that does not use data values. So there exists some deterministic word automaton $\mathcal{A}_\pi = (Q_\pi, q_\pi, F_\pi, \delta_\pi)$ such that for every (data)word w, if $\mathcal{A}_\pi$ starts reading w at position y, $\mathcal{A}_\pi$ is in a final state every time it reaches a position x such that $(x, y) \in [\![\pi]\!]_w$. As we also asked that π be non-ambiguous, for all

positions x, there is at most one position y such that starting the automaton $\mathcal{A}_\pi$ at position y leads to x being in a final state. Similarly, there exists some deterministic word automaton $\mathcal{A}_{nr} = (Q_{nr}, q_{nr}, F_{nr}, \delta_{nr})$ that, if started at the initial position of the word, is in a final state at position x if and only if a position x has *no reachable* destination with π, meaning that there is actually no position y such that $(x, y) \in [\![\pi]\!]_w$, which also implies that, wherever $\mathcal{A}_\pi$ is started, it will never reach x with a final state. For example, if $\phi_i = \langle\varepsilon\rangle < \langle a?\cdot \leftarrow^2\rangle$, then $\mathcal{A}_{nr}$ recognize the first two positions (because we cannot go left twice from them) and all positions where the letter is not a.
Obviously ϕ_i cannot hold in these positions, so we can modify $\mathcal{A}_{nr}$ a bit to reject (by forcing a transition to $\bot$) the words where c_i is read when in a final state. This takes care of not reachable positions, so it remains to check whether the c_i markings are correct for reachable positions.
From $\mathcal{A}_\pi$, we first define an A1RA $\mathcal{A}_r$ that will check if ϕ_i holds for all positions x that can be reached from the position y from which $\mathcal{A}_r$ has been started. Let $\mathcal{A}_r = (Q_\pi, q_\pi, Q_\pi, \Delta_r)$ where:

- Q_π, q_π are the same as in $\mathcal{A}_\pi$, and all states are final.
- $\Delta_r = \{(q, (a, s), \emptyset, (\delta_\pi(q, a), \emptyset)) \mid \text{if } q \notin F_\pi\}$
  $\cup \{(q, (a, s), \bowtie, (\delta_\pi(q, a), \emptyset)) \mid \text{if } q \in F_\pi \text{ and } c_i \in s\}$
  $\cup \{(q, (a, s), \bar{\bowtie}, \bot) \mid \text{if } q \in F_\pi \text{ and } c_i \in s\}$
  $\cup \{(q, (a, s), \bar{\bowtie}, (\delta_\pi(q, a), \emptyset)) \mid \text{if } q \in F_\pi \text{ and } c_i \notin s\}$
  $\cup \{(q, (a, s), \bowtie, \bot) \mid \text{if } q \in F_\pi \text{ and } c_i \notin s\}$

for all $(a, s) \in \Sigma \times 2^C$, and where $\bar{\bowtie}$ is the comparison operator opposite to $\bowtie$ (i.e. $\bar{<}$ is $\geq$, $\bar{\neq}$ is $=$, etc).
Then let $\mathcal{A}'_i = (Q, q_0, F, \Delta)$ be an A1RA, whose role consists in launching $\mathcal{A}_r$ simultaneously at all positions, be defined as:

- $Q = \{q, q'\} \uplus Q_\pi$
- $q_0 = q$
- $F = Q$
- $\Delta = \Delta_r$
  $\cup \{(q, \varepsilon, \emptyset, (q_\pi, \downarrow) \wedge (q', \emptyset))\}$[1]
  $\cup \{(q', (a, s), \emptyset, (q, \emptyset))\}$

And finally, $\mathcal{A}_i$ is the conjunction of $\mathcal{A}'_i$ and $\mathcal{A}_{nr}$.

Proposition 6. *$\mathcal{A}_i$ accepts w' if and only if the markings of c_i are correct.*

Proof. The only way a computation can fail is by following a transition that leads to $\bot$, as it can easily be checked that in any case, some transition is available. These transitions happen in two cases: either $\mathcal{A}_{nr}$ found that the current position is not reachable and claims to satisfy c_i, which is a contradiction ; or $\mathcal{A}_r$ has been started from some position y (from which it kept the value in the

[1] We allow ourselves ε-transition to simplify a bit the description, but those can be eliminated using the usual way

register), and later at position x, $\mathcal{A}_r$ is in a state that is final for $\mathcal{A}_\pi$, which means that y is the only position such that $(x, y) \in [\![\pi]\!]_w$, and then either x claims to satisfy ϕ_i (c_i is marked) but the test with the register fails (or alternatively, the opposite test $\bar{\bowtie}$ succeeded), or it claims not to satisfy ϕ_i (c_i is not marked) but the test actually succeeded, both of which are contradictions. Conversely, these are the only possible ways of contradicting the markings. Thus, $\mathcal{A}_i$ fails if and only if there is a contradiction in the markings. □

The case of right-oriented paths is a bit different.
Let $\phi_i = \langle\varepsilon\rangle \bowtie \langle\pi^{\rightarrow}\rangle$. Again, π is a right-oriented PDL path that does not use data values, so let $\mathcal{A}_\pi = (Q_\pi, q_\pi, F_\pi, \delta_\pi)$ be a deterministic automaton recognizing π. This time, we have to define two different A1RA to check when c_i holds and when it does not. Let $\mathcal{A}_i^+ = (Q^+, q^+, F^+, \Delta^+)$ where:

- $Q^+ = \{q^+\} \cup Q_\pi$
- $F^+ = \emptyset$
- $\Delta^+ = \{q^+, (a, s), \emptyset, \top \mid \text{if } c_i \notin s\}$
  $\cup \{(q^+, (a, s), \bowtie, \top) \mid \text{if } c_i \in s \text{ and } q_\pi \in F_\pi\}$
  $\cup \{(q^+, (a, s), \emptyset, (\delta_\pi(q_\pi, a), \emptyset)) \mid \text{if } c_i \in s\}$
  $\cup \{(q, (a, s), \bowtie, \top) \mid \forall q \in F_\pi\}$
  $\cup \{(q, (a, s), \emptyset, (\delta_\pi(q, a), \emptyset)) \mid \forall q \in Q_\pi\}$

Let $\mathcal{A}_i^- = (Q^-, q^-, F^-, \Delta^-)$ where:

- $Q^- = \{q^-\} \cup Q_\pi$
- $F^- = Q^-$
- $\Delta^- = \{q^-, (a, s), \emptyset, \top \mid \text{if } c_i \in s\}$
  $\cup \{(q^-, (a, s), \bowtie, \bot) \mid \text{if } c_i \notin s \text{ and } q_\pi \in F_\pi\}$
  $\cup \{(q^-, (a, s), \bar{\bowtie}, (\delta_\pi(q_\pi, a), \emptyset)) \mid \text{if } c_i \notin s \text{ and } q_\pi \in F_\pi\}$
  $\cup \{(q^-, (a, s), \emptyset, (\delta_\pi(q_\pi, a), \emptyset)) \mid \text{if } c_i \notin s \text{ and } q_\pi \notin F_\pi\}$
  $\cup \{(q, (a, s), \bowtie, \bot) \mid \forall q \in F_\pi\}$
  $\cup \{(q, (a, s), \bar{\bowtie}, (\delta_\pi(q, a), \emptyset)) \mid \forall q \in F_\pi\}$
  $\cup \{(q, (a, s), \emptyset, (\delta_\pi(q, a), \emptyset)) \mid \forall q \in Q_\pi \backslash F_\pi\}$

In both automata, q^+ (resp. q^-) play the role of q_π and also checks that c_i is marked (resp. not marked), otherwise it accepts because it is the other automaton's job to check this. This is why the transition with q^+ (q^-) and the following transitions with $q \in Q_\pi$ look similar.

Lemma 1. *For any w, if $\mathcal{A}_i^+$ (resp. $\mathcal{A}_i^-$) is launched at position x, then it does not accept if and only if c_i is marked (resp. not marked) and this marking is incorrect.*

Proof. $\mathcal{A}_i^+$ fails if and only if there is no witness y such that $(x, y) \in \pi$ and $w.d[x] \bowtie w.d[y]$, i.e. ϕ_i does not hold. Conversely, $\mathcal{A}_i^-$ fails if and only if there is such a witness. □

Now we define $\mathcal{A}_i = (Q, q_0, F, \Delta)$ whose job is simply to launch $\mathcal{A}_i^+$ and $\mathcal{A}_i^-$ at each position:

– $Q = \{q_0, q_1\} \uplus Q^+ \uplus Q^-$
– $F = \{q_0, q_1\} \uplus F^-$
– $\Delta = \Delta^+ \uplus \Delta^-$
$\cup \{(q_0, \varepsilon, \emptyset, (q_1, \emptyset) \wedge (q^+, \downarrow) \wedge (q^-, \downarrow))\}$
$\cup \{(q_1, (a, s), \emptyset, (q_0, \emptyset))\}$

Proposition 7. *$\mathcal{A}_i$ accepts w if and only if the markings of c_i are correct.*

Proof. Consequence of Lemma 1.

Note that as we reduce to the emptiness of A1RA, this decision procedure has non-elementary complexity.

4.2 Returning to the general case

We added two restrictions to simplify the construction:

– the left hand side path is ε
– there are no nested comparison tests

We will now explain the intuition for how to deal with the general case.

Let $\phi_i = \langle \pi \rangle \bowtie \langle \pi' \rangle$ be a comparison test where π is not restricted to ε, however the other restrictions (no nested comparison tests, one direction only, non-ambiguous if left-oriented) still apply.

Proposition 8. *One can effectively build an A1RA $\mathcal{A}_i$ which accepts a dataword if and only if the c_i markings are correct.*

Proof. There are four different cases depending on the orientation of π and π':

The cases $\langle \pi^{\leftarrow} \rangle \bowtie \langle \pi'^{\rightarrow} \rangle$ and $\langle \pi^{\rightarrow} \rangle \bowtie \langle \pi'^{\leftarrow} \rangle$ are symmetrical, so let us assume that π is the left-oriented path. In order to check this kind of test, we compose the two constructions defined earlier: the automaton starts by lauching $\mathcal{A}_{nr}$ and simultaneously launches $\mathcal{A}_r$ at every position, as before. However, instead of making tests when ending in a final state, $\mathcal{A}_r$ simply branches to $\mathcal{A}^+$ and $\mathcal{A}^-$. This is done by replacing any transition $p \to q$ where $q \in F_\pi$ by a transition $p \to q \wedge q^+ \wedge q^-$ with the same letter. The only time a value is stored in the register is at the beginning of $\mathcal{A}_r$, so the comparisons in $\mathcal{A}^+$ and $\mathcal{A}^-$ will refer to this value.

In the case $\langle \pi^{\leftarrow} \rangle \bowtie \langle \pi'^{\leftarrow} \rangle$, the idea is to start $(\mathcal{A}_\pi, \mathcal{A}_{\pi'})$ at every couple (y, y') of positions in the word. In other words, for every positions $y < y'$ in w, there is a branch where the automaton started $\mathcal{A}_\pi$ at position y and later on started $\mathcal{A}_{\pi'}$ at position y', and vice-versa. The data value is stored when the first automaton is started, and a comparison is done when the second starts. If $\bowtie$ holds and $\mathcal{A}_\pi$ was started before $\mathcal{A}_{\pi'}$, then we reject only if both are in a final state, and at that position c_i is not marked. Conversely, if $\not\bowtie$ holds, then we reject if both are in a final state and c_i is marked. If $\mathcal{A}_{\pi'}$ was started before

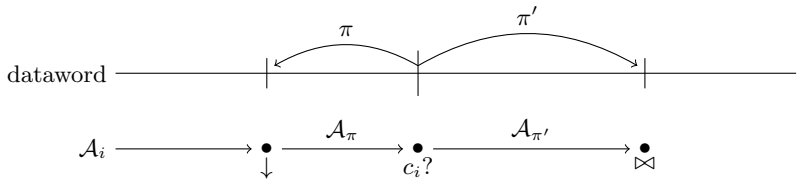

Fig. 1. An illustration for the case left-right

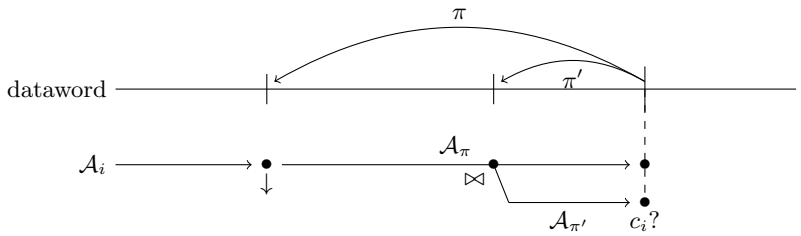

Fig. 2. An illustration for the case left-left

$\mathcal{A}_\pi$, we just need to take care of reversing the direction of the comparison, i.e. $<$ becomes $>$, $\leq$ becomes $\geq$ and vice-versa.

Sadly the case $\langle \pi^{\rightarrow} \rangle \bowtie \langle \pi'^{\rightarrow} \rangle$ is a bit more difficult, as the paths may be ambiguous, which means there are potentially many "candidate" couples of positions. At every position, we start both $\mathcal{A}_\pi$ and $\mathcal{A}_{\pi'}$, and we remember whether c_i was marked at that position or not.
In the case where c_i was marked, whenever any of the two automata reaches a final state, it can non-deterministically stop or continue. If it chose to stop, it stores the current value, then waits for the second automaton to reach a final state and non-deterministically choose to stop, at which point a comparison is made according to which automaton stopped first (if $\mathcal{A}_\pi$ stopped first, the comparison is $\bowtie$, else it's the reversed one). The computation accepts only if both automata chose to stop at some point and the comparison test held (the two positions where the automata stopped give a witness proving ϕ_i).
In the case where c_i was not marked, the construction is a bit similar: when any of the two automata reaches a final state, it both co-non-deterministically stops and continues. In the branch where the automaton stopped, it stores the current value, waits for a branch where the second automaton stops, and then tests whether $\bowtie$ holds (or the reversed comparison depending on who stopped first). This time, a computation fails only if there is a branch where both automata stopped and the comparison test held (which gives a witness proving ϕ_i while c_i was not marked).

□

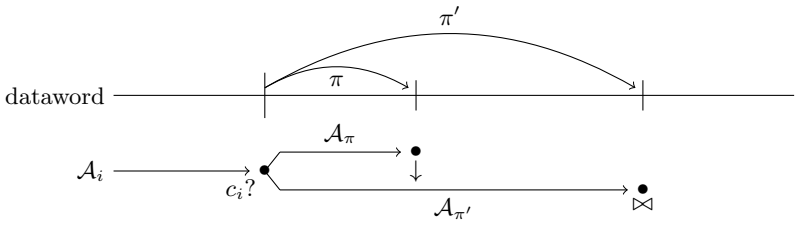

Fig. 3. An illustration for the case right-right

Now let us consider a comparison test subformula in which paths may contain comparison tests themselves. As we have seen how to deal with the case when paths do not contain nested comparison tests, we can recursively convert every nested test into a c_i so that every ϕ_i is a path containing no comparison tests. For instance, the formula $\langle\varepsilon\rangle < \langle\rightarrow \cdot\{\langle\varepsilon\rangle > \langle\rightarrow^*\rangle\}?\cdot \rightarrow\rangle$ is divided into two markings:

- First c_0 corresponds to $\phi_0 = \langle\varepsilon\rangle > \langle\rightarrow^*\rangle$
- Then c_1 corresponds to $\phi_1 = \langle\varepsilon\rangle < \langle\rightarrow \cdot\{c_0\}?\cdot \rightarrow\rangle$

with the corresponding $\mathcal{A}_0$ and $\mathcal{A}_1$. The rest of the construction remain unchanged.

5 A separation theorem for DataPDL

In the DataPDL$^{\ominus}$ fragment, we ask that paths in comparison tests must be restricted to a single direction. This may seem very restrictive, but we show that actually every path can be transformed into several paths each using a single direction. The intuition is that a path without comparison tests is equivalent to a two-way automaton, which we will "split" into multiple one-way automata using the usual *crossing sequences* construction.

Proposition 9. *A formula $\langle\pi\rangle \bowtie \langle\pi'\rangle$ where π and π' contain no comparison tests is equivalent to a disjunction of formulas of the form $\langle\pi_i\rangle \bowtie \langle\pi'_i\rangle \wedge \phi_i$ where π_i and π'_i are paths using only one direction, and ϕ_i is a formula without comparison tests.*

Proof. Let $\phi = \langle\pi\rangle \bowtie \langle\pi'\rangle$ be such a formula. As π and π' are data-free paths, they are equivalent to some two-way automaton which we will call $\mathcal{A}$ (for π) and $\mathcal{B}$ (for π'). These automata start at current position x, move in the dataword, and *accept* some positions, i.e. reach some final state at these positions. Without loss of generality, suppose they each have a single initial state $q_0^{\mathcal{A}}$ and $q_0^{\mathcal{B}}$ and a single final state $q_f^{\mathcal{A}}$ and $q_f^{\mathcal{B}}$. Let us focus on $\mathcal{A}$ for the moment.
Suppose that position y is accepted by $\mathcal{A}$, and let us take some shortest path leading to q_f at position y. This path gives a sequence of states $q_0 \cdot q_1 \cdots q_n \cdot q_f$,

where each state is also annotated with the direction (left or right) in which the word was read at that moment. A position z visited in this path may have been visited several times, so let $\mathcal{C}_z$ be the *crossing sequence* of z, that is, the sequence $q_{i_1} \cdots q_{i_k}$ of states in which $\mathcal{A}$ crossed position z, as well as the direction they were following. The same state with the same direction cannot appear twice in $\mathcal{C}_z$, otherwise that would imply the existence of a shorter accepting path. Thus, crossing sequences have size at most $2 \cdot |Q|$ (and so there is a finite number of them). Let us call $\mathcal{C}_0$ the crossing sequence of x and $\mathcal{C}_f$ the one of y. $\mathcal{C}_0$ starts with q_0, and has odd length if and only if q_0 is directed towards y. Similarly, $\mathcal{C}_f$ ends with q_f, and has odd length if and only if q_f is directed away from x. For all positions z strictly between x and y, $\mathcal{C}_z$ has odd length, and for all other positions, $\mathcal{C}_z$ has even length.
We can verify that the crossing sequences of two consecutive positions *match* each other by checking a few local conditions involving the two crossing sequences and the letter at their respective positions. Intuitively, these conditions only check that there exists some transitions in $\mathcal{A}$ which are used to get from one side to the other.
The one-way automata we will build has these crossing sequences as state space, starts from some initial crossing sequence $\mathcal{C}_0$, and simply guesses the sucessive crossing sequences while checking they match, until it reaches a final crossing sequence $\mathcal{C}_f$, thus only one pass will be necessary. Suppose that $\mathcal{C}_0$ and $\mathcal{C}_f$ are fixed. There are two symmetric cases, either $\mathcal{C}_0$ is to the left of position $\mathcal{C}_f$, or to the right. For instance, suppose it is the first case. Then we build three automata:

- $\mathcal{A}^{\rightarrow}(\mathcal{C}_0, \mathcal{C}_f)$ that only moves forward, starting at state $\mathcal{C}_0$, guessing matching crossing sequences, until it reaches $\mathcal{C}_f$ then stops.
- $\mathcal{A}^{\rightarrow}(\mathcal{C}_f)$ that also only moves forward, starting at $\mathcal{C}_f$, and ending on the empty crossing sequence.
- $\mathcal{A}^{\leftarrow}(\mathcal{C}_0)$ that only moves backward, starting at $\mathcal{C}_0$, and also ending on the empty crossing sequence.

The first automaton guesses the part of the path that is between $\mathcal{C}_0$ and $\mathcal{C}_f$, while the two other respectively guess the parts of the path that are after $\mathcal{C}_f$ and before $\mathcal{C}_0$. Put together, these three guessed parts recreate the original path followed by $\mathcal{A}$. In the case where the respective position of $\mathcal{C}_0$ and $\mathcal{C}_f$ are reversed, the three automata simply have their direction reversed. With this, we have that for any formula ψ, $\langle \mathcal{A} \rangle \psi$ is equivalent to:

$$\bigvee_{\substack{\mathcal{C}_0, \mathcal{C}_f \\ d \in \{\leftarrow, \rightarrow\}}} \left(\langle \mathcal{A}^d(\mathcal{C}_0, \mathcal{C}_f) \cdot \{ \langle \mathcal{A}^d(\mathcal{C}_f) \rangle \top \}? \rangle \psi \wedge \langle \mathcal{A}^{\bar{d}}(\mathcal{C}_0) \rangle \top \right)$$

that is, we simply guess $\mathcal{C}_0$, $\mathcal{C}_f$, and the direction d (we note $\bar{d}$ the opposite direction), then we guess the three parts of the path, and the formula ψ is tested at the end of the part between $\mathcal{C}_0$ and $\mathcal{C}_f$. We do the same construction for the other two-way automaton $\mathcal{B}$. Put together, we can now translate the original

comparison test $\phi = \langle \pi \rangle \bowtie \langle \pi' \rangle$:

$$\begin{aligned}
&\phi \rightsquigarrow \langle \mathcal{A} \rangle \bowtie \langle \mathcal{B} \rangle \\
&\rightsquigarrow \bigvee_{\substack{\mathcal{C}_0, \mathcal{C}_f, \mathcal{C}'_0, \mathcal{C}'_f, \\ d, d' \in \{\leftarrow, \rightarrow\}}} \Big(\langle \mathcal{A}^d(\mathcal{C}_0, \mathcal{C}_f) \cdot \{\langle \mathcal{A}^d(\mathcal{C}_f) \rangle \top\}? \rangle \bowtie \langle \mathcal{B}^{d'}(\mathcal{C}'_0, \mathcal{C}'_f) \cdot \{\langle \mathcal{B}^{d'}(\mathcal{C}'_f) \rangle \top\}? \rangle \\
&\qquad\qquad \wedge \Big(\langle \mathcal{A}^{\bar{d}}(\mathcal{C}_0) \rangle \top \wedge \langle \mathcal{B}^{\bar{d}'}(\mathcal{C}'_0) \rangle \top \Big) \Big)
\end{aligned}$$

All these automata are one-way, so they can be translated back into a PDL path using only one direction. □

6 Conclusion and discussion

We introduced a new logic on datawords of which a fragment is decidable. This fragment is expressive enough to express some properties such as "data are increasing" or "there is a leader with maximal value", while also being able to express regular properties such as "the word has even size", which could be of interest for model-checking.
Some possible axes for future research include extending our results to the settings of infinite datawords as well as datatrees, in order to increase the number of behaviours that can be modeled. Infinite datawords may model the execution of a reactive program, a program that always reacts to inputs from the environment, while datatrees may simulate branching programs.

References

1. Cyriac Aiswarya, Benedikt Bollig, and Paul Gastin. An automata-theoretic approach to the verification of distributed algorithms. *arXiv preprint arXiv:1504.06534*, 2015.
2. Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic (TOCL)*, 12(4):27, 2011.
3. Christopher Czyba, Christopher Spinrath, and Wolfgang Thomas. Finite automata over infinite alphabets: Two models with transitions for local change. In *International Conference on Developments in Language Theory*, pages 203–214. Springer, 2015.
4. Stéphane Demri and Ranko Lazić. Ltl with the freeze quantifier and register automata. *ACM Transactions on Computational Logic (TOCL)*, 10(3):16, 2009.
5. Diego Figueira. A decidable two-way logic on data words. In *Logic in Computer Science (LICS), 2011 26th Annual IEEE Symposium on*, pages 365–374. IEEE, 2011.
6. Diego Figueira, Piotr Hofman, and Sławomir Lasota. Relating timed and register automata. *Mathematical Structures in Computer Science*, pages 1–29, 2010.
7. Michael J Fischer and Richard E Ladner. Propositional dynamic logic of regular programs. *Journal of computer and system sciences*, 18(2):194–211, 1979.

8. Erich Grädel and Martin Otto. On logics with two variables. *Theoretical computer science*, 224(1):73–113, 1999.
9. Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
10. Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic (TOCL)*, 9(2):10, 2008.
11. Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic (TOCL)*, 5(3):403–435, 2004.
12. Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. In *International Workshop on Computer Science Logic*, pages 499–513. Springer, 2010.