Pebble-Intervals Automata and FO^2 with Two Orders

Nadia Labai^(✉), Tomer Kotek, Magdalena Ortiz, and Helmut Veith

TU Wien, Vienna, Austria
labai@dbai.tuwien.ac.at

Abstract. We introduce a novel automata model, which we call *pebble-intervals automata (PIA)*, and study its power and closure properties. PIAs are tailored for a decidable fragment of FO that is important for reasoning about structures that use data values from infinite domains: the two-variable fragment with one total preorder and its induced successor relation, one linear order, and an arbitrary number of unary relations. We prove that the string projection of every language of data words definable in the logic is accepted by a pebble-intervals automaton $\mathcal{A}$, and obtain as a corollary an automata-theoretic proof of the EXPSPACE upper bound for finite satisfiability due to Schwentick and Zeume.

Keywords: Automata and logic · Pebble-intervals automata · Emptiness testing · Decidability · Two-variable fragment · Data words

1 Introduction

Finding decidable fragments of First Order Logic (FO) that are expressive enough for reasoning in different applications is a major line of research. A prominent such fragment is *the two-variable fragment FO^2 of FO* , which has a decidable finite satisfiability problem [13, 21] and is well-suited for handling graph-like structures. It captures many *description logics*, which are prominent formalisms for knowledge representation, and several authors have recently applied fragments based on FO^2 to verification of programs [1, 7, 8, 16, 26]. Unfortunately, FO^2 has severe limitations, e.g., it cannot express transitivity, and in the applications to verification above, it cannot reason about programs whose variables range over data values from infinite domains. This has motivated the exploration of decidable extensions of FO^2 with special relations which are not axiomatizable in FO^2 . For example, finite satisfiability of FO^2 with a linear order was shown to be NEXPTIME -complete in [24], even in the presence of the induced successor relation [12], and equivalence relations have been used to

N. Labai—This work was supported by the Austrian Science Fund (FWF) projects P30360, P30873, and W1255.

This article is dedicated to the memory of Helmut Veith, who passed away tragically while this manuscript was being prepared.

© Springer Nature Switzerland AG 2020

A. Leporati et al. (Eds.): LATA 2020, LNCS 12038, pp. 208–221, 2020.

https://doi.org/10.1007/978-3-030-40608-0_14

model data values which can be tested for equality [3, 4, 10, 23]. However, related extensions of FO^2 with preorders easily become undecidable [3, 19]. Recently the logic $\text{FO}^2(\leq_1, \lesssim_2, S_2)$, that is FO^2 with a linear order $\leq_1$, a total preorder $\lesssim_2$ and its induced successor S_2 , and any number of unary relations from a finite alphabet, was shown to have an EXPSPACE-complete satisfiability problem [27]. This logic can compare data values in terms of which is smaller than which and whether they are consecutive in $\lesssim_2$, making it suitable to model linearly ordered data domains, and a good candidate for extending existing verification methods which use two-variable logics. We continue the study of $\text{FO}^2(\leq_1, \lesssim_2, S_2)$, and in particular, focus on a suitable automata model for it. Establishing a connection to suitable automata for fragments of FO that can talk about values from infinite domains is an active area of research. Automata are also important in automated verification, where they are used, for example, to reason about temporal properties of program traces [9, 30]. We make the following contributions:

- As an automata model for $\text{FO}^2(\leq_1, \lesssim_2, S_2)$ we propose *pebble-intervals automata* (PIAs). Similarly to classical finite-state automata, PIAs are read-once automata for strings. However, they read the input in varying order. Using a fixed set of pebbles $[m] = \{1, \dots, m\}$, a PIA reads a position p by choosing three pebbles $i, j, k \in [m]$ and non-deterministically moving k to position p between the positions of i and j .
- We study the computational power and closure properties of PIAs. We describe a restricted class of PIAs that accept exactly the regular languages, and show that some context-free languages, and even languages which are not context-free, are accepted by PIAs. We prove that PIAs are effectively closed under union, concatenation, Kleene star, shuffle, and iterated shuffle, but not effectively closed under intersection, even with regular languages, nor under complement.
- We show that the emptiness problem for PIA is NL-complete if the number of pebbles is logarithmic in the size of the automaton, and is PSPACE in general.
- We show that PIAs contain $\text{FO}^2(\leq_1, \lesssim_2, S_2)$ in the following sense: for each sentence ψ , there is a PIA whose language coincides with the *projection language* of ψ , obtained by omitting $\lesssim_2$ and S_2 from the structures satisfying ψ .
- As a corollary, we get an automata-theoretic proof for EXPSPACE membership of finite satisfiability for $\text{FO}^2(\leq_1, \lesssim_2, S_2)$ that was established in [27].

Due to space limitations, we have omitted the proofs from the body of the paper. They can be found in the extended version.

2 Pebble-Intervals Automata

In this section, we introduce pebble-intervals automata (PIA). We study their emptiness problem, their expressive power, and closure properties of the languages they accept.

Let $[n] = \{1, \dots, n\}$. A *string* of length $n \geq 0$ over alphabet Σ is a mapping $w : [n] \rightarrow \Sigma$, written also $w = w(1) \cdots w(n)$. Note that $[0] = \emptyset$ and $w : [0] \rightarrow \Sigma$ is the empty string ε . We often use s, u, v , and w for strings, and $|w|$ for the length of w .

A PIA is equipped with a finite number m of pebbles. It begins its computation with no pebbles on the input w , and uses MOVE transitions to place and replace pebbles. In a $k\text{-MOVE}_{i,j}$ transition, the pebble k (which may or may not have been previously placed on w) is non-deterministically placed on a previously unread position in the interval between pebbles i and j . The input boundaries can be used as interval boundaries, e.g., a $k\text{-MOVE}_{i,\triangleleft}$ transition places pebble k to the right of pebble i . For convenience we allow *silent* transitions that go to a new state without moving any pebbles. As pebbles can only be placed on unread positions, each position of w is read at most once. In an accepting run all positions must be read, and the run must end at an accepting state.

Definition 1 (Pebble-intervals automata). A PIA $\mathcal{A}$ is a tuple $(\Sigma, m, Q, q_{\text{init}}, F, \delta)$, where Σ is the (finite) alphabet, $m \in \mathbb{N}$, Q is the finite set of states, $q_{\text{init}} \in Q$ is the initial state, $F \subseteq Q$ are the accepting states, and $\delta \subseteq (Q \times Q) \cup (Q \times \text{MOVE}_m \times \Sigma \times Q)$ is the transition relation with $\text{MOVE}_m = \{k\text{-MOVE}_{i,j} \mid i \in [m] \cup \{\triangleright\}, j \in [m] \cup \{\triangleleft\}, k \in [m], i \neq j\}$. We may omit m when it is clear from the context. Transitions in $Q \times \text{MOVE} \times \Sigma \times Q$ are MOVE transitions, and transitions in $Q \times Q$ are silent transitions. The size of $\mathcal{A}$ is $|\delta| + |\Sigma| + |Q|$.

The positions of m pebbles on a string of length n during a run of a PIA are described by an (m, n) -pebble assignment, which is a function $\tau : [m] \rightarrow [n] \cup \{\perp\}$ with either $\tau(i) \neq \tau(j)$ or $\tau(i) = \tau(j) = \perp$ for each $1 \leq i < j \leq m$; the pebbles j with $\tau(j) = \perp$ are unassigned. We define $\rho_{\perp}$ as $\rho_{\perp}(i) = \perp$ for every $i \in [m]$. By $\hat{\tau} : [m] \cup \{\triangleright, \triangleleft\} \rightarrow \{0\} \cup [n+1]$ we denote the extension of τ with $\hat{\tau}(\triangleright) = 0$ and $\hat{\tau}(\triangleleft) = n+1$.

Definition 2 (Semantics of PIAs). Consider a PIA $\mathcal{A} = (\Sigma, m, Q, q_{\text{init}}, F, \delta)$. A configuration of $\mathcal{A}$ on string $u \in \Sigma^*$ is a triple (q, ρ, N) where $q \in Q$ is the current state, $\rho : [m] \rightarrow [|u|] \cup \{\perp\}$ is the current pebble assignment, and $N \subseteq [|u|]$ is the set of already-read positions. The initial configuration π_{init} is $(q_{\text{init}}, \rho_{\perp}, \emptyset)$. A configuration (q, ρ, N) is accepting if $q \in F$ and $N = [|u|]$. Let $\pi = (q, \rho, N)$ and $\pi' = (q', \rho', N')$ be configurations on u . We call them consecutive and write $\pi \xrightarrow{t} \pi'$ if there exists a transition t in δ such that either:

1. t is a silent transition of the form (q, q') , $N = N'$, and $\rho = \rho'$; or
2. t is a MOVE transition $(q, k\text{-MOVE}_{i,j}, u(\ell), q')$ with $\hat{\rho}(i) < \ell < \hat{\rho}(j)$ and $\ell \in [|u|] - N$, and additionally $\rho' = \rho[k \mapsto \ell]$ and $N' = N \cup \{\ell\}$. That is, pebble k is placed on position ℓ in the open interval between i and j , reading the letter $u(\ell)$.

Let $\bar{t} = (t_1, \dots, t_r)$ and $\bar{\pi} = (\pi_0, \dots, \pi_r)$ be sequences of transitions and configurations. We call $(\bar{t}, \bar{\pi})$ a computation of $\mathcal{A}$ on u if $\pi_0 = \pi_{\text{init}}$ and $\pi_{i-1} \xrightarrow{t_i} \pi_i$.

π_i for every $i \in [r]$, and write $\pi_0 \xrightarrow{\bar{t}} \pi_r$. We call $(\bar{t}, \bar{\pi})$ accepting if π_r is accepting. We write $\pi \xrightarrow{*} \pi'$ if $\pi \xrightarrow{\bar{t}} \pi'$ for some $\bar{t}$. The automaton $\mathcal{A}$ accepts u if there is an accepting computation of $\mathcal{A}$ on u . The set of all u accepted by $\mathcal{A}$ is denoted $L(\mathcal{A})$, and called a PI language.

Computational Power of Pebble-Intervals Automata. PIAs generalize standard non-deterministic finite-state automata. A PIA $\mathcal{A} = (\Sigma, 1, Q, q_{\text{init}}, F, \delta)$ with one pebble is *unidirectional* if q_{init} has no incoming transitions, and the MOVE transitions from other states use $1\text{-MOVE}_{1, \triangleleft}$ only.

Proposition 1. *A language L is accepted by a standard non-deterministic finite-state automaton iff $L = L(\mathcal{A})$ for a unidirectional PIA $\mathcal{A}$ with the same number of states.*

PI languages also contain non-regular languages, and even some non-context-free ones.

Examples 1. *The following are examples of PI languages:*

1. *There is a PIA $\mathcal{A}_{\text{Dyck}}$ with one pebble that accepts the Dyck language L_{Dyck} of well-nested brackets, which is context-free but not regular. The alphabet has two letters $[$ and $]$, and the states are q_l and q_r . The initial and only accepting state is q_l . The transition relation contains $(q_l, 1\text{-MOVE}_{\triangleright, \triangleleft}, [, q_l)$ and $(q_l, 1\text{-MOVE}_{1, \triangleleft},], q_l)$. $\mathcal{A}_{\text{Dyck}}$ accepts a string iff there are as many left as right brackets, and no prefix has more right than left brackets.*
2. *A similar one pebble PIA accepts the language L_{two} of all strings of two types of parentheses, where each type is well-nested with respect to itself, but not necessarily to the other type. E.g., $([]) \in L_{\text{two}}$, but $() \notin L_{\text{two}}$. L_{two} is not context-free.*
3. *$\{a^n \$ b^n \# c^n \mid n \geq 0\}$, which is not context-free, is accepted by a PIA with 3 pebbles. Pebbles 1 and 2 read the $\$$ and the $\#$, and then the PIA keeps doing the following: pebble 3 reads an a to the left of pebble 1, a b between pebbles 1, 2, and a c to the right of pebble 2.*
4. *$\{w\$w \mid w \in \{0, 1\}^+\}$ is not context-free, and is accepted by a PIA with 3 pebbles. Pebble 1 reads the $\$$, pebble 2 reads a letter σ to the left of pebble 1, and pebble 3 also reads σ to the right of pebble 1. Then the PIA repeats: (i) a letter σ is non-deterministically chosen, (ii) pebble 2 reads σ between its current position and pebble 1, and (iii) pebble 3 reads σ to the right of its current position. Similar languages are PI languages, e.g., $\{ww^R ww \mid w \in \Sigma^*\}$, where w^R is w in reverse.*

We conjecture that not all context-free language are PI languages; e.g, the Dyck language of two types of well-nested parentheses seems not to be PI.

Closure Properties. We provide a construction of suitable PIAs in the appendix to show the following.

Theorem 1. *The class of PI languages is effectively closed under union, concatenation, Kleene- $\star$, shuffle, and iterated shuffle. It is not effectively closed under intersection, even with regular languages, nor under complement.*

From the construction used in the proof of the above theorem, we also obtain:

Corollary 1. *The universality and inclusion problems for PIAs are undecidable.*

Emptiness. For deciding whether $L(\mathcal{A}) \neq \emptyset$ for a given PIA, we use *feasible sequences of transitions*, which are those that correspond to an actual computation of a PIA. One can show that for a given PIA with m pebbles, $L(\mathcal{A}) \neq \emptyset$ iff there is a feasible sequence of transitions $\bar{t}$ of length at most $|\mathcal{A}| \cdot 2^{O(m \log m)}$, and that the existence of the latter can be guessed and verified using a bounded amount of information (roughly a counter, two transitions, and two pebble assignments). This gives us the upper bounds below, which hold also if $\mathcal{A}$ is not given explicitly, as long as δ can be computed non-deterministically in $\log(|\mathcal{A}|)$ space. For the case where $\mathcal{A}$ has $O(\log |\mathcal{A}|)$ pebbles, NL-hardness follows from the same result for standard finite state automata and Proposition 1.

Theorem 2. *If a PIA $\mathcal{A}$ has $O(\log |\mathcal{A}|)$ pebbles, its emptiness problem is NL-complete. In general, the emptiness problem for PIA is in PSPACE.*

Related Automata Models. *Jumping finite automata* [20] are probably the closest to PIAs: they are essentially PIAs with one pebble, which is placed on an arbitrary unvisited position without specifying an interval. In the context of languages with infinite alphabets, various automata models have been proposed that run on *data words*: string words where values from an infinite domain are attached to each position. Register automata are finite-state machines on data words which use registers to compare whether data values are equal [6, 17, 22]; their string projection languages are regular. Pebble automata [22] use pebbles in a stack discipline to test for equality of data values. Data automata [3–5] are an extension of register automata introduced to prove the decidability of satisfiability of FO^2 on words with a linear order, a successor relation, and an equivalence relation. Their projection languages are accepted by multicounter automata, which are finite automata on strings extended with counters, that are equivalent to Vector Addition Systems or Petri Nets [11]. Class Memory Automata [2] have the same expressive power as data automata. Variable Finite Automata [15] extend finite state automata with variables from an infinite alphabet. Many works have studied these automata models and their variations, see [28] and [18, Chapter 4] for surveys.

3 PIAs and $\text{FO}^2(\leq_1, \lesssim_2, S_2)$

To establish the relation between $\text{FO}^2(\leq_1, \lesssim_2, S_2)$ and PIAs, we need some preliminaries. Recall that a total preorder $\lesssim$ is a transitive total relation which can

be seen as an equivalence relation whose equivalence classes are linearly ordered. We use $x \sim_2 y$ as shorthand for $(x \preceq_2 y) \wedge (y \preceq_2 x)$. The induced successor relation S of a total preorder $\preceq$ is such that $S(x, y)$ if $x \preceq y$ and there is no z such that $x \preceq z \preceq y$.

Two-variable logic (FO^2) is the restriction of FO to formulas that only use two variables x and y , and $\text{FO}^2(\leq_1, \preceq_2, S_2)$ is FO^2 with a linear order $\leq_1$, a total preorder $\preceq_2$ and its induced successor S_2 , and any number of unary relations from a finite alphabet.

All structures and strings in this paper are finite. For a structure $\mathcal{A}$, we denote its universe by A and its size by $|A|$. The *empty structure* has $A = \emptyset$ and is denoted $\emptyset_{\text{voc}}$.

Data Words. Let Σ a finite alphabet. Its *extension for data words* is $\text{voc}_{\text{DW}}(\Sigma) = \langle \leq_1, \preceq_2, S_2, \sigma : \sigma \in \Sigma \rangle$. A *data word* over Σ is a finite $\text{voc}_{\text{DW}}(\Sigma)$ -structure $\mathcal{D}$ with universe D where $\sigma : \sigma \in \Sigma$ are interpreted as unary relations that partition D . We use $\mathcal{D}$, $\mathcal{D}'$, etc. to denote data words. The empty word is denoted by $\emptyset_{\text{DW}(\Sigma)}$, and the class of all data words over Σ by $\text{DW}(\Sigma)$. A set of data words is called a *data language*.

Let φ_1, φ_2 be $\text{FO}^2(\text{voc}_{\text{DW}}(\Sigma))$ formulas. We write $\varphi_1 \models_{\text{DW}(\Sigma)} \varphi_2$ if $\mathcal{D} \models \varphi_1$ implies $\mathcal{D} \models \varphi_2$ for every $\mathcal{D} \in \text{DW}(\Sigma)$, and define equivalence $\equiv_{\text{DW}(\Sigma)}$ analogously. We may omit Σ if clear from context. The *data value* $\text{value}_{\mathcal{D}}(d)$ of an element $d \in D$ is the number of equivalence classes E of $\sim_2$ whose elements $d' \in E$ satisfy $d' \preceq_2 d$, and $\text{maxval}_{\mathcal{D}} = \max_{d \in D} \text{value}_{\mathcal{D}}(d)$. The *string projection* of $\mathcal{D}$, denoted $\text{string}(\mathcal{D})$, is the string w of length $|w| = |D|$ where for all $\ell \in [|w|]$, $w(\ell) = \sigma$ if and only if $\mathcal{D} \models \sigma(d)$ where d is the unique element of $\mathcal{D}$ such that $\ell = |\{d' \in D \mid \mathcal{D} \models d' \leq_1 d\}|$. The projection of the empty structure $\emptyset_{\text{voc}_{\text{DW}}(\Sigma)}$, and only of $\emptyset_{\text{voc}_{\text{DW}}(\Sigma)}$, is ε . The *projection language* of a data language Δ is the string language $L(\Delta) = \{w \mid w = \text{string}(\mathcal{D}) \text{ for some } \mathcal{D} \in \Delta\}$. If a formula ψ defines Δ , we write $L(\psi)$ for $L(\Delta)$.

Example 1. To avoid ambiguity, in our running examples we use underlined symbols. Let $\Xi = \{\xi_1, \xi_2\}$ be a set of unary relations and let $\underline{\mathcal{D}}$ be the data word with universe $\underline{D} = \{\underline{a}, \underline{b}, \underline{c}, \underline{d}, \underline{e}, \underline{f}\}$ where $\leq_1$ is the lexicographic order, the interpretation of ξ_1 is $\{\underline{a}, \underline{b}, \underline{c}, \underline{e}\}$, the interpretation of ξ_2 is $\{\underline{d}, \underline{f}\}$, and $\underline{b} \preceq_2 \underline{a} \preceq_2 \underline{e} \preceq_2 \underline{c} \preceq_2 \underline{d} \sim_2 \underline{f}$. Note e.g. that $\underline{\mathcal{D}} \models S_2(\underline{a}, \underline{e})$ and $\underline{\mathcal{D}} \models \neg S_2(\underline{b}, \underline{e}) \wedge (\underline{b} \preceq_2 \underline{e})$. The string projection of $\underline{\mathcal{D}}$ is $\text{string}(\underline{\mathcal{D}}) = \xi_1 \xi_1 \xi_1 \xi_2 \xi_1 \xi_2$.

The goal of this section is to prove the following theorem.

Theorem 3. *If ψ is a $\text{FO}^2(\leq_1, \preceq_2, S_2)$ sentence, there is a PIA $\mathcal{A}$ with $L(\psi) = L(\mathcal{A})$.*

To prove this, we rely on the normal form defined next. A *1-type* $\nu(x)$ over a vocabulary $\text{voc}_{\text{DW}}(\Sigma)$ is a maximal consistent conjunction of atomic and negated atomic formulas with the free variable x . A *2-type* $\theta(x, y)$ is defined similarly. Given a $\text{FO}^2(\text{voc}_{\text{DW}}(\Sigma))$ formula ψ , we obtain a φ in normal form by taking the Scott Normal Form [14, Theorem 2.1] of ψ , and expanding the quantifier-free formulas to Disjunctive Normal Form, and in fact to disjunctions of 2-types θ .

The Scott Normal Form of ψ introduces linearly many new symbols, resulting in an extended Σ' . We let $\Xi = \{\xi_a \mid a \in [A]\}$ be an alphabet containing a symbol for every 1-type over Σ' .

Theorem 4 (Normal Form). *Let $\psi \in \text{FO}^2(\text{voc}_{\text{DW}}(\Sigma))$. Then there exist $A, B, C \in \mathbb{N}$, an alphabet $\Xi = \{\xi_a \mid a \in [A]\}$, a formula $\varphi \in \text{FO}^2(\text{voc}_{\text{DW}}(\Xi))$ of the form $\varphi = \varphi_{\forall} \wedge \varphi_{\exists}$ and a letter-to-letter substitution $h : \Xi \rightarrow \Sigma$ such that $L(\psi) = h(L(\varphi))$,*

$$\varphi_{\forall} = \forall x \forall y \bigvee_{\theta \in \Theta_{\forall}} \theta(x, y) \quad \varphi_{\exists} = \varphi_{\varepsilon} \wedge \forall x \bigwedge_{a \in [A]} \xi_a(x) \rightarrow \bigwedge_{b \in [B]} \exists y \bigvee_{c \in [C]} \theta_{abc}(x, y)$$

with θ and θ_{abc} 2-types over $\text{voc}_{\text{DW}}(\Xi)$, and $\varphi_{\varepsilon} = \text{True}$ if $\emptyset_{\text{voc}_{\text{DW}}(\Sigma)} \models_{\text{DW}(\Sigma)} \psi$ and $\varphi_{\varepsilon} = \exists x(\text{True})$ if $\emptyset_{\text{voc}_{\text{DW}}(\Sigma)} \not\models_{\text{DW}(\Sigma)} \psi$. Moreover, φ is computable in EXPSPACE and is of length exponential in $|\psi|$.

We let $\Theta_{\exists} = \{\theta_{abc} \mid a \in [A], b \in [B], c \in [C]\}$ and $\Theta = \Theta_{\forall} \cup \Theta_{\exists}$. Given $a \in [A]$, a *witness type set* for a is a choice of 2-types satisfying the right-hand side of the implication for ξ_a . That is, a set of 2-types $\omega \subseteq \Theta_{\exists}$ that contains one θ_{abc} for every $b \in [B]$, representing a choice of the existential constraints an element needs to fulfill. Denote by Ω_a the set of witness type sets for a and let $\Omega = \bigcup_{a \in [A]} \Omega_a$. For a witness type set $\omega \in \Omega$, let $\omega(x) = \bigwedge_{\theta \in \omega} \exists y \theta(x, y)$ be its *existential constraints*. Note that $\omega(x)$ is always satisfiable and that there is a unique letter $\xi^{\omega} \in \Xi$ such that $\omega(x) \models_{\text{DW}(\Xi)} \xi^{\omega}(x)$.

Example 2. Consider the following formula φ given in normal form

$$\forall x \forall y \chi(x, y) \wedge \forall x (\underline{\xi}_1(x) \rightarrow \exists y (\underline{\theta}_1(x, y) \vee \underline{\theta}_3(x, y)) \wedge \underline{\xi}_2(x) \rightarrow \exists y (\underline{\theta}_2(x, y) \vee \underline{\theta}_4(x, y)))$$

where $\chi(x, y)$ is the disjunction of 2-types equivalent to $(\underline{\xi}_2(x) \wedge \underline{\xi}_2(y)) \rightarrow x \sim_2 y$, and the $\underline{\theta}_i$ are given as the following 2-types (omitted clauses are negated):

$$\begin{array}{ll} \underline{\theta}_1 = x <_1 y \wedge S_2(x, y) \wedge \underline{\xi}_1(x) \wedge \underline{\xi}_2(y) & \underline{\theta}_3 = x <_1 y \wedge \neg S_2(x, y) \wedge x \lesssim_2 y \wedge \underline{\xi}_1(x) \wedge \underline{\xi}_2(y) \\ \underline{\theta}_2 = y <_1 x \wedge S_2(y, x) \wedge \underline{\xi}_2(x) \wedge \underline{\xi}_1(y) & \underline{\theta}_4 = y <_1 x \wedge \neg S_2(y, x) \wedge y \lesssim_2 x \wedge \underline{\xi}_2(x) \wedge \underline{\xi}_1(y) \end{array}$$

A data word satisfies φ iff it is the empty structure, or (a) the largest element of $\leq_1$ has letter $\underline{\xi}_2$, (b) the smallest element of $\leq_1$ has letter $\underline{\xi}_1$, (c) all elements with $\underline{\xi}_2$ have maximal value, and (d) no element with $\underline{\xi}_1$ has maximal value.

Note that $\underline{D} \models \varphi$. The projection language $L(\varphi)$ is the regular language with regular expression $\underline{\xi}_1(\underline{\xi}_1 + \underline{\xi}_2)^* \underline{\xi}_2 + \varepsilon$. We have $\Theta_{\exists} = \{\underline{\theta}_1, \underline{\theta}_2, \underline{\theta}_3, \underline{\theta}_4\}$. For φ , we have $A = 2$, $B = 1$, and $C = 2$. The witness type sets of φ are $\{\underline{\theta}_{111}\}$, $\{\underline{\theta}_{112}\}$, $\{\underline{\theta}_{211}\}$, and $\{\underline{\theta}_{212}\}$, where $\underline{\theta}_{111} = \underline{\theta}_1$, $\underline{\theta}_{112} = \underline{\theta}_3$, $\underline{\theta}_{211} = \underline{\theta}_2$, and $\underline{\theta}_{212} = \underline{\theta}_4$. Hence, we have $\underline{\Omega} = \{\{\underline{\theta}_1\}, \{\underline{\theta}_2\}, \{\underline{\theta}_3\}, \{\underline{\theta}_4\}\}$, and $\xi^{\{\underline{\theta}_1\}} = \xi^{\{\underline{\theta}_3\}} = \underline{\xi}_1$, and $\xi^{\{\underline{\theta}_2\}} = \xi^{\{\underline{\theta}_4\}} = \underline{\xi}_2$.

We construct a PIA $\mathcal{A}^{\varphi}$ that accepts a string w iff it can be extended into a data word $\mathcal{D}$ that satisfies the normal form φ of a given sentence ψ . Note that ψ and φ have different alphabets, but since there is a letter-to-letter substitution h such that $L(\psi) = h(L(\mathcal{A}^{\varphi}))$, and PIAs are closed under letter-to-letter substitutions, this proves Theorem 3.

For constructing our PIA, we first focus on the existential part, i.e., whether w can be extended into a $\mathcal{D}$ that satisfies $\varphi_\exists$. This is achieved in two steps: **(S1)** We reduce the existence of $\mathcal{D}$ to the existence of a sequence of *consecutive task words*, data words that store additional information of already satisfied vs. ‘promised’ subformulas; the sequence should lead to a *completed* task word where all promises are fulfilled. **(S2)** We do not have a bound on the length of task words and their data values, so we use *extremal strings* to decide the existence of the desired sequence with the limited memory of PIAs. After these two steps, we introduce *perfect* extremal strings to guarantee the satisfaction of $\varphi_\forall$. Our PIA will then decide if a sequence of perfect extremal strings exists.

Task Words for $\varphi_\exists$. We start by defining *task words*, which are like data words but do more book-keeping. Additionally to data values, elements in task words are assigned *tasks*, which are witness type sets where each 2-type may be marked as *completed* if its satisfaction has already been established, or as *promised* otherwise. We reduce the satisfaction of $\varphi_\exists$ to the existence of a sequence of $\mathcal{T}_1, \dots, \mathcal{T}_n$ of *consecutive* task words, where we keep assigning new data values and updating promised into completed tasks, until we reached a *completed* task word $\mathcal{T}_n$.

Definition 3 (Tasks). For $\theta \in \Theta_\exists$, we call C_θ a completed task and P_θ a promised task. Let $\text{Tasks}_C = \{C_\theta \mid \theta \in \Theta_\exists\}$, $\text{Tasks}_P = \{P_\theta \mid \theta \in \Theta_\exists\}$ and $\text{Tasks} = \text{Tasks}_C \cup \text{Tasks}_P$.

For each task set $ts \subseteq \text{Tasks}$, there is at most one witness type set $\omega \in \Omega$ that ts realizes, which means that for every $\theta \in \Theta_\exists$, **(1)** $|\{P_\theta, C_\theta\} \cap ts| \leq 1$, and **(2)** $|\{P_\theta, C_\theta\} \cap ts| = 1$ if and only if $\theta \in \omega$. If there is such an ω , we denote it $\omega(ts)$, and call ts an Ω -realization. The set of all Ω -realizations is 2_Ω^{Tasks} , and $2_\Omega^{\text{Tasks}_C} = 2_\Omega^{\text{Tasks}} \cap 2^{\text{Tasks}_C}$ and $2_\Omega^{\text{Tasks}_P} = 2_\Omega^{\text{Tasks}} \cap 2^{\text{Tasks}_P}$.

Example 3. Since the witness type sets in $\underline{\Omega}$ are singletons, so are the $ts \in 2_\Omega^{\text{Tasks}}$. Let $\underline{ts}_i^C = \{C_{\theta_i}\}$ and $\underline{ts}_i^P = \{P_{\theta_i}\}$ for $i \in [4]$. Then we have $2_\Omega^{\text{Tasks}} = \{\underline{ts}_i^C \mid i \in [4]\} \cup \{\underline{ts}_i^P \mid i \in [4]\}$, and $\{C_{\theta_i}\}$ and $\{P_{\theta_i}\}$ are $\{\theta_i\}$ -realizations for $i \in [4]$.

$\mathcal{D}$ -task words are data words that assign tasks to the elements of $\mathcal{D}$. More precisely, each $d \in D$ is assigned, instead of a letter ξ_a , a task set ts that realizes a witness type set ω which contains C_θ for each $\theta \in \omega$ that d satisfies, and P_θ for the remaining $\theta \in \omega$.

Definition 4 (Task word). Let $\mathcal{D}$ be a data word over Ξ . A $\mathcal{D}$ -task word is a data word $\mathcal{T}$ over 2_Ω^{Tasks} that has the same universe and order relations as $\mathcal{D}$, and for every $d \in D$ with $\mathcal{T} \models ts(d)$, **(1)** $\mathcal{D} \models \xi^{\omega(ts)}(d)$, and **(2)** for every $\theta \in \omega(ts)$, $C_\theta \in ts$ iff $\mathcal{D} \models \exists y \theta(d, y)$. A task word $\mathcal{T}$ is a $\mathcal{D}$ -task word for some $\mathcal{D}$, and it is completed if $\mathcal{T} \models \varphi_\varepsilon \wedge \forall x \bigvee_{ts \in 2_\Omega^{\text{Tasks}_C}} ts(x)$.

Example 4. We define a $\underline{\mathcal{D}}$ -task word $\underline{\mathcal{T}}$; its vocabulary is 2_Ω^{Tasks} , its universe is $\{a, b, c, d, e, f\}$, and $\leq_1, \lesssim_2$, and S_2 are the same as in $\underline{\mathcal{D}}$. The interpretation of the letter $\underline{ts}_1^C$ is $\{c\}$, that of $\underline{ts}_2^C$ is $\{d\}$, that of $\underline{ts}_3^C$ is $\{a, b, e\}$, and that of

ts_4^C is $\{f\}$; the other letters are empty. As $\underline{D} \models \varphi$, all existential constraints are satisfied and $\underline{T}$ is completed.

The satisfaction of $\varphi_{\exists}$ coincides with the existence of a completed task word.

Lemma 1. *Let $\mathcal{D} \in \text{DW}(\Xi)$. There exists a completed $\mathcal{D}$ -task word iff $\mathcal{D} \models \varphi_{\exists}$.*

We now characterize the notion of *consecutive* task words using *trimmings*.

Definition 5 (Trimming, consecutiveness). *The trimming of a data word $\mathcal{D}$, denoted $\mathcal{D}^{\setminus 1}$, is the substructure of $\mathcal{D}$ induced by removing the elements with the maximal data value. For task words, trimmings are obtained by removing the elements with the largest data value and updating the tasks of the remaining elements correctly. That is, a trimming of a $\mathcal{D}$ -task word $\mathcal{T}$ is a $\mathcal{D}^{\setminus 1}$ -task word $\mathcal{T}_1$ such that $\omega(ts) = \omega(ts_1)$ for every d and every $ts, ts_1 \in 2_{\Omega}^{\text{Tasks}}$ with $\mathcal{T} \models ts(d)$ and $\mathcal{T}_1 \models ts_1(d)$. We say that $\mathcal{T}_1, \mathcal{T}$ are consecutive if $\mathcal{T}_1$ is a trimming of $\mathcal{T}$.*

The trimming of a task word is unique, and we denote it $\mathcal{T}^{\setminus 1}$.

Example 5. $\underline{D}^{\setminus 1}$ is obtained from $\underline{D}$ by removing $\underline{d}$ and $\underline{f}$. The $\underline{D}^{\setminus 1}$ -task word $\underline{T}^{\setminus 1}$ has universe $\{\underline{a}, \underline{b}, \underline{c}, \underline{e}\}$ and order relations as in $\underline{D}^{\setminus 1}$. Note that $\underline{d}$ and $\underline{f}$ contributed in $\underline{D}$ to the satisfaction of $\varphi_{\exists}$, so $\underline{T}^{\setminus 1}$ has promised tasks and is no longer completed, with interpretations $\underline{ts}_1^P = \{\underline{c}\}$, $\underline{ts}_3^P = \{\underline{a}, \underline{b}, \underline{e}\}$, and the remaining letters empty. Note that the tasks for the shared elements of $\underline{T}$ and $\underline{T}^{\setminus 1}$ realize the same witness type sets.

We have achieved **(S1)**: reducing satisfaction of $\varphi_{\exists}$ to finding a sequence of task words.

Proposition 2. *There exists a data word $\mathcal{D} \models \varphi_{\exists}$ if and only if there is a sequence $\mathcal{T}_1 \dots, \mathcal{T}_n$ of consecutive task words, where $\mathcal{T}_n$ is a completed $\mathcal{D}$ -task word.*

Now to **(S2)**: as the limited memory of PIAs hinders the manipulation of task words with unbounded length and data values, we operate on their *extremal strings* instead.

First, in *data abstractions* of task words, we do not distinguish all data values, but only the *top layer* elements with maximal value, the *second to top layer*, and the rest. We let $\text{Layers} = \{1\text{top}, 2\text{top}, \text{rest}\}$, and define the alphabet $\Gamma = \text{Layers} \times 2_{\Omega}^{\text{Tasks}}$. We also define its restrictions to completed and promised tasks as $\Gamma_C = \text{Layers} \times 2_{\Omega}^{\text{Tasksc}}$ and $\Gamma_P = \text{Layers} \times 2_{\Omega}^{\text{Tasksp}}$, while $\Gamma_h = \{h\} \times 2_{\Omega}^{\text{Tasks}}$ is the restriction of Γ to some specific $h \in \text{Layers}$. For a symbol $\gamma = (h, ts)$ in Γ , we denote $ts(\gamma) = ts$ and $\omega(\gamma) = \omega(ts)$.

Definition 6 (Data abstraction). *Let $\mathcal{T}$ be a $\mathcal{D}$ -task word. For every $d \in D$, let ts_d be such that $\mathcal{T} \models ts_d(d)$, and let $\mathcal{A}$ be the data word over Γ with same universe and order relations as $\mathcal{T}$, and with $\mathcal{A} \models \gamma_h(d)$ where $\gamma_h = (h, ts_d)$ iff (a) $h = 1\text{top}$ and $\text{value}_{\mathcal{D}}(d) = \text{maxval}_{\mathcal{D}}$, (b) $h = 2\text{top}$ and $\text{value}_{\mathcal{D}}(d) = \text{maxval}_{\mathcal{D}} - 1$, or (c) $h = \text{rest}$ and $\text{value}_{\mathcal{D}}(d) \in [\text{maxval}_{\mathcal{D}} - 2]$. The data abstraction $\text{abst}(\mathcal{T})$ of $\mathcal{T}$ is the string projection $\text{string}(\mathcal{A})$.*

Extremal strings are obtained from data abstractions by keeping only the maximal and minimal positions in each layer with respect to the tasks. We extend to them the notions of *consecutive* and *completed*.

Definition 7 (extremal strings). For $w \in \Gamma^*$, define its extremal positions $\text{extPos}(w)$:

$$\begin{aligned} \text{pos}_{h,\theta}(w) &= \{\ell \in [|w|] \mid w(\ell) = (h, ts), \theta \in \omega(ts)\} \\ \text{pos}_{\text{rest}, P_\theta}(w) &= \{\ell \in [|w|] \mid w(\ell) = (\text{rest}, ts), P_\theta \in ts\} \\ \text{extPos}_{h,\theta}(w) &= \{\ell \mid \ell = \max(\text{pos}_{h,\theta}(w)) \text{ or } \ell = \min(\text{pos}_{h,\theta}(w))\} \\ \text{If } \theta \models x \leq_1 y, \quad \text{extPos}_\theta(w) &= \{\ell \mid \ell = \max(\text{pos}_{\text{rest}, P_\theta}(w))\} \\ \text{If } \theta \models y <_1 x, \quad \text{extPos}_\theta(w) &= \{\ell \mid \ell = \min(\text{pos}_{\text{rest}, P_\theta}(w))\} \\ \text{extPos}(w) &= \bigcup_{\theta \in \Theta_\exists} (\text{extPos}_\theta(w) \bigcup_{h \in \text{Layers}} \text{extPos}_{h,\theta}(w)) \end{aligned}$$

If $\text{extPos}(w) = \{\ell_1, \dots, \ell_r\}$ and $\ell_1 < \dots < \ell_r$, then the extremal string of w is $\text{ext}(w) = w(\ell_1) \dots w(\ell_r)$. $\text{EXT}(\Gamma) = \{\text{ext}(w) \mid w \in \Gamma^*\}$ denotes the set of extremal strings. Note that $s = \text{ext}(w)$ implies $\text{ext}(s) = s$ and $\text{ext}(\varepsilon) = \varepsilon$. An extremal string s is *completed* if $s \in \Gamma_C^+$, or if $s = \varepsilon$ and $\emptyset_{\text{DW}(\Sigma)} \models_{\text{DW}(\Sigma)} \psi$. We occasionally write $\text{ext}(\mathcal{T})$ to mean $\text{ext}(\text{abst}(\mathcal{T}))$.

A pair s', s of extremal strings is *consecutive* if $s' = \text{ext}(\mathcal{T}^{\setminus 1})$ and $s = \text{ext}(\mathcal{T})$ for some task word $\mathcal{T}$.

For an extremal string s and $\ell \in [|s|]$, the set of letters that can augment s at position ℓ without being extremal is $\Gamma^{\text{not ext}}(s, \ell) = \{\gamma \in \Gamma \mid \text{ext}(s) = \text{ext}(s(1) \dots s(\ell-1)\gamma s(\ell) \dots s(|s|))\}$, and we define $\Gamma_h^{\text{not ext}}(s, \ell) = \Gamma_h \cap \Gamma^{\text{not ext}}(s, \ell)$ for $h \in \text{Layers}$.

Example 6. Let $\underline{w}$ be the following 6-letter string over $\underline{\Gamma} = \text{Layers} \times 2^{\underline{\Omega}^{\text{Tasks}}}$:

$$(\text{rest}, \underline{ts}_3^C)(\text{rest}, \underline{ts}_3^C)(2\text{top}, \underline{ts}_1^C)(1\text{top}, \underline{ts}_2^C)(\text{rest}, \underline{ts}_3^C)(1\text{top}, \underline{ts}_4^C).$$

Then $\text{extPos}(\underline{w}) = \{1, 3, 4, 5, 6\}$, since the letter at position 2 appears both to the left, at position 1, and to the right, at position 5, and $\underline{s} = \text{ext}(\underline{w})$ is the substring obtained from $\underline{w}$ by removing the non-extremal position 2.

This concludes (S2), reducing the existence of $\mathcal{D}$ to a sequence of extremal strings.

Corollary 2. *There is a data word $\mathcal{D} \models \varphi_\exists$ if and only if there is a sequence of consecutive extremal strings where the last one is completed.*

Perfect Extremal Strings for $\varphi_\forall$. We define in the appendix a formula which intuitively ‘extracts’ the 2-type of elements in a data word. Let $\alpha = (h_\alpha, ts_\alpha)$ and $\beta = (h_\beta, ts_\beta)$ in Γ with at least one of them in $\Gamma_{1\text{top}}$. The formula $\text{perf}_{\alpha,\beta}(x, y)$ implies for every atomic formula either itself or its negation. For example, if $h_\alpha = h_\beta = 1\text{top}$, then $\text{perf}_{\alpha,\beta,\lesssim_2}(x, y)$ implies $x \lesssim_2 y$, $y \lesssim_2 x$, $\neg S_2(x, y)$, and $\neg S_2(y, x)$. Hence for all $\alpha, \beta \in \Gamma$ with at least one of them in $\Gamma_{1\text{top}}$, there exists a 2-type $\theta(x, y)$ such that $\text{perf}_{\alpha,\beta}(x, y) \equiv_{\text{DW}(\Xi)} \theta(x, y)$. This allows us to describe the 2-type of elements in task words via $\text{perf}_{\alpha,\beta}$ formulas. For any two

elements of the data word, there is a (possibly iterated) trimming in which both appear and one of them has the maximal data value, and their perfect formula, which is equivalent to their 2-type, determines whether they satisfy the universal constraint χ . Thus we can ensure satisfaction of χ using $\text{perf}_{\alpha,\beta}(x, y)$ formulas from all the trimmings.

Definition 8 (Perfect string, perfect task word). *Let $w \in \Gamma^*$. We say w is a perfect string if for every two positions $\ell_1 < \ell_2$ in w such that $\{w(\ell_1), w(\ell_2)\} \cap \Gamma_{1\text{top}} \neq \emptyset$ we have $\text{perf}_{w(\ell_1), w(\ell_2)}(x, y) \models_{\text{DW}(\Xi)} \chi(x, y) \wedge \chi(y, x)$. Note that the empty string ε is perfect. A task word $\mathcal{T}$ is perfect if it is empty, or if $\text{ext}(\mathcal{T})$ and $\text{ext}(\mathcal{T}^{\mathbb{N}^1})$ are perfect.*

Example 7. Let $\underline{\alpha} = (2\text{top}, \underline{ts}_1^C)$ and $\underline{\beta} = (1\text{top}, \underline{ts}_2^C)$. Then $\text{perf}_{\underline{\alpha}, \underline{\beta}}(x, y)$ is given by: $\text{perf}_{\underline{\alpha}, \underline{\beta}}(x, y) = \underline{\xi}_1(x) \wedge \underline{\xi}_2(y) \wedge (x <_1 y) \wedge (x \prec_2 y) \wedge S_2(x, y)$.

The 2-type θ to which $\text{perf}_{\underline{\alpha}, \underline{\beta}}(x, y)$ is equivalent over $\text{DW}(\Xi)$ is given by the conjunction of $\text{perf}_{\underline{\alpha}, \underline{\beta}}(x, y)$ with $\neg \underline{\xi}_2(x) \wedge \neg \underline{\xi}_1(y) \wedge (y \not<_1 x) \wedge (y \not\prec_2 x) \wedge \neg S_2(y, x)$. We have that $\underline{w}$ is a perfect string, and $\text{perf}_{\underline{\alpha}, \underline{\beta}}(x, y) \models_{\text{DW}(\Xi)} \underline{\chi}(x, y) \wedge \underline{\chi}(y, x)$.

We characterize satisfiability in terms of perfect completed task words.

Lemma 2. *Let $\mathcal{T}$ be a $\mathcal{D}$ -task word. $\mathcal{T}$ is perfect if and only if $\mathcal{D} \models \varphi_{\forall}$.*

As a corollary of Lemma 2 and Lemma 1, we get:

Proposition 3. *For every data word $\mathcal{D} \in \text{DW}(\Xi)$, $\mathcal{D} \models \varphi$ if and only if there exists a perfect completed $\mathcal{D}$ -task word. There is $\mathcal{D} \models \varphi$ if and only if there is a sequence of consecutive perfect extremal strings where the last one is completed.*

We are almost ready to define $\mathcal{A}^\varphi$. Intuitively, it will guess a sequence of extremal strings as in Proposition 3, placing pebbles from an extremal string to a consecutive one. This requires the automaton to verify consecutiveness, and to know which positions in consecutive extremal strings correspond to the same position in the input. This is easy if we have the underlying task word; indeed, given a task word $\mathcal{T}$ and an extremal string $s' = \text{ext}(\mathcal{T})$, there is a bijective mapping from the *extremal elements* of $\mathcal{T}$ that s' stores, to their positions in s' . The same holds for $\mathcal{T}^{\mathbb{N}^1}$ and $s = \text{ext}(\mathcal{T}^{\mathbb{N}^1})$. By composing these mappings after inverting the latter, and restricting its domain to positions that remain extremal after updating the abstracted data values (that is, shifting the top layer to second top, and the second top into the remaining layer), we obtain a *partial embedding from s to s' via $\mathcal{T}$* that keeps track of the matching positions; the precise definition is in the appendix. But one major hurdle remains: these notions are defined in terms of a task word $\mathcal{T}$, and our PIA cannot store task words, only their extremal strings. We overcome this through a merely syntactic characterization of consecutiveness, which can be verified without a concrete task word. This rather technical step relies on the fact that if s, s' are consecutive, then s' can be obtained by guessing a substring r that will get new data values, interleaving it into the proper positions g of s , which can also be guessed, and updating the abstracted data values. Also the partial embedding that keeps track of matching the positions can be obtained without a concrete $\mathcal{T}$, using r and g .

Lemma 3. *We can decide whether two given extremal strings s, s' are consecutive in EXPSPACE . If they are, then we can also obtain in EXPSPACE a partial embedding $\text{PEmb}_{s \hookrightarrow s'}$ from positions in s to positions in s' that coincides with the partial embedding from s to s' via T for every task word T such that $s' = \text{ext}(T)$ and $s = \text{ext}(T^{\setminus 1})$.*

The Automaton. We give a high-level description of $\mathcal{A}^\varphi = (\Xi, m + 1, Q, q_{\text{init}}, F, \delta)$, and refer to the extended version for a full definition. We have $m = 7 \cdot |\Theta_\exists|$: there is one pebble for each existential constraint in $\Theta_\exists$ and each layer in Γ , plus an additional pebble per constraint, and one designated pebble $m + 1$ to read non-extremal positions. $Q = Q_e \cup Q_p$ has two types of states:

- Q_e contains states (s, τ) with s a perfect extremal string and τ an $(m + 1, |s|)$ -pebble assignment, which intuitively describes the assignment after reading s .
- Q_p contains states of the forms $(s, \tilde{s}, \tau, 0)$ and $(s, \tilde{s}, \tau, 1)$ for every perfect extremal string s , non-empty prefix $\tilde{s}$ of s , and $(m + 1, |s|)$ -pebble assignment τ that satisfies certain conditions that hold when only the prefix $\tilde{s}$ has been read.

The initial state is $q_{\text{init}} = (\varepsilon, \rho_\perp) \in Q_e$ and the final states are $F = \{(s, \tau) \in Q_e \mid s \text{ is completed}\}$. The transition δ is roughly as follows. $\mathcal{A}^\varphi$ should transition from $(s, \tau) \in Q_e$ to $(s', \tau') \in Q_e$ for consecutive s, s' , but since it can only move one pebble at a time, we have intermediate states in Q_p which allow it to read s' from left to right by iterating over all its prefixes. We start reading s' by moving to $(s', s'(1), \tau'_{q'}, 0) \in Q_p$, where $\tau'_{q'}$ stores the pebble assignment induced by $\text{PEmb}_{s' \hookrightarrow s}$. Once the whole extremal string s' has been read, we move to the next extremal state.

This finishes the construction of the automaton $\mathcal{A}^\varphi$ with $L(\mathcal{A}^\varphi) = L(\varphi)$, and thus the proof of Theorem 3. Concerning the upper bound on finite satisfiability, by Theorem 4 and $\text{EXT}(\Gamma) \subseteq \Gamma^{7|\Theta_\exists|}$, we get that $\mathcal{A}^\varphi$ has size at most double exponential in $|\psi|$. For the EXPSPACE upper bound, we need to show that the transition relation of $\mathcal{A}^\varphi$ is EXPSPACE -computable (Lemma 3 in the appendix). This with Theorem 2 gives an alternative proof of the upper bound in [27]:

Corollary 3. *Finite satisfiability of $\text{FO}^2(\leq_1, \lesssim_2, S_2)$ is in EXPSPACE .*

Relation to the Proof of Schwentick and Zeume [27]. Naturally, there are similarities between the techniques; our extremal strings and tasks are similar to their profiles and directional constraints. However, a key difference is that in their ‘geometric’ view, elements of the data word are assigned points (a, b) in the plane with a a position in $\leq_1$, and b a data value. Existential constraints are indicated by marking the witnesses with the letters they should have, and many profiles in a consistent sequence can contain points with the same a value. In contrast, our ‘temporal’ view arises from the computation of the PIA. We mark elements with existential constraints they need to satisfy and that they have satisfied, which is compatible with the read-once nature of PIA. It does not seem possible to use their proof techniques without modifying PIA to allow

multiple readings of the input. The modified model would work for the logic-to-automata relation established here, but we suspect it would be too strong for the other direction.

4 Discussion and Conclusion

We introduced pebble-intervals automata (PIA) and studied their computational power. We proved that the projections of data languages definable in $\text{FO}^2(\leq_1, \preceq_2, S_2)$ are PI languages, and as a by-product, obtained an alternative proof that finite satisfiability is in EXPSpace . The main question that remains is the converse of our main result: whether every PI language is the projection of an $\text{FO}^2(\leq_1, \preceq_2, S_2)$ definable data language. We believe this is the case. Our work also gives rise to other questions. We suspect that our results can be extended to ω -languages, and we would like to adapt them to C^2 , which extends FO^2 with counting quantifiers [25, 29]. We also plan to explore further the computational power of our automata model, for instance, to establish a pumping lemma that allows us to prove that some context-free languages are not PI languages.

Acknowledgments. We thank the anonymous reviewers for their helpful remarks.

References

1. Ahmetaj, S., Calvanese, D., Ortiz, M., Simkus, M.: Managing change in graph-structured data using description logics. *ACM Trans. Comput. Logic* **18**(4), 27:1–27:35 (2017)
2. Björklund, H., Schwentick, T.: On notions of regularity for data languages. *Theor. Comput. Sci.* **411**(4–5), 702–715 (2010)
3. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. In: *PODS*, pp. 10–19. ACM (2006)
4. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data words. *TOCL* **12**(4), 27 (2011)
5. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: *LICS*, pp. 7–16. IEEE (2006)
6. Bouyer, P., Petit, A., Thérien, D.: An algebraic approach to data languages and timed languages. *Inf. Comput.* **182**(2), 137–162 (2003)
7. Calvanese, D., Kotek, T., Šimkus, M., Veith, H., Zuleger, F.: Shape and content. In: Albert, E., Sekerinski, E. (eds.) *IFM 2014. LNCS*, vol. 8739, pp. 3–17. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10181-1_1
8. Calvanese, D., Ortiz, M., Simkus, M.: Verification of evolving graph-structured data under expressive path constraints. In: *ICDT*, pp. 15:1–15:19 (2016)
9. Clarke, E., Henzinger, T.A., Veith, H., Bloem, R. (eds.): *Handbook of Model Checking*. Springer, Cham (2018). <https://doi.org/10.1007/978-3-319-10575-8>
10. David, C., Libkin, L., Tan, T.: On the satisfiability of two-variable logic over data words. In: Fermüller, C.G., Voronkov, A. (eds.) *LPAR 2010. LNCS*, vol. 6397, pp. 248–262. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16242-8_18

11. Esparza, J.: Decidability and complexity of Petri net problems — an introduction. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_20
12. Etessami, K., Vardi, M., Wilke, T.: First-order logic with two variables and unary temporal logic. *Inform. Comput.* **179**(2), 279–295 (2002)
13. Grädel, E., Kolaitis, P., Vardi, M.: On the decision problem for two-variable first-order logic. *Bull. Symb. Log.* **3**(01), 53–69 (1997)
14. Grädel, E., Otto, M.: On logics with two variables. *Theor. Comput. Sci.* **224**(1), 73–113 (1999)
15. Grumberg, O., Kupferman, O., Sheinvald, S.: Variable automata over infinite alphabets. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 561–572. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13089-2_47
16. Itzhaky, S., et al.: On the automated verification of web applications with embedded SQL. In: ICDT. LIPIcs, vol. 68, pp. 16:1–16:18 (2017)
17. Kaminski, M., Francez, N.: Finite-memory automata. *Theor. Comput. Sci.* **134**(2), 329–363 (1994)
18. Kara, A.: Logics on data words. Ph.D. thesis, Technical University of Dortmund (2016)
19. Manuel, A., Zeume, T.: Two-variable logic on 2-dimensional structures. In: CSL, pp. 484–499 (2013)
20. Meduna, A., Zemek, P.: Jumping finite automata. *Int. J. Found. Comput. Sci.* **23**(7), 1555–1578 (2012)
21. Mortimer, M.: On languages with two variables. *Math. Log. Q.* **21**(1), 135–140 (1975)
22. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. *TOCL* **5**(3), 403–435 (2004)
23. Niewerth, M., Schwentick, T.: Two-variable logic and key constraints on data words. In: ICDT, pp. 138–149. ACM (2011)
24. Otto, M.: Two variable first-order logic over ordered domains. *J. Symb. Logic* **66**(2), 685–702 (2001)
25. Pratt-Hartmann, I.: Logics with counting and equivalence. In: CSL-LICS, pp. 76:1–76:10. ACM (2014)
26. Rensink, A.: Canonical graph shapes. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 401–415. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24725-8_28
27. Schwentick, T., Zeume, T.: Two-variable logic with two order relations. *Log. Meth. Comput. Sci.* **8**(1) (2012)
28. Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 41–57. Springer, Heidelberg (2006). https://doi.org/10.1007/11874683_3
29. Tan, T.: Extending two-variable logic on data trees with order on data values and its automata. *ACM Trans. Comput. Log.* **15**(1), 8:1–8:39 (2014)
30. Vardi, M., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS, pp. 332–344. IEEE (1986)