Two-variable logic and Key Constraints on Data Words[*]

Matthias Niewerth
matthias.niewerth@udo.edu

Thomas Schwentick
thomas.schwentick@udo.edu

TU Dortmund University

ABSTRACT

The paper introduces key constraints for data words and shows that it is decidable whether, for a given two-variable sentence φ that can refer to the successor relation on positions and a set $\mathcal{K}$ of key constraints, there is a data string w that satisfies φ and respects $\mathcal{K}$. Here, the formula is allowed to refer to the successor relation but not to the linear order on the positions of the word. As a byproduct, a self-contained exposition of an algorithm that decides satisfiability of such formulas (without key constraints) in 2-NEXPTIME is given.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data Models*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic

General Terms

Algorithms, Theory

1. INTRODUCTION

It is well known that two-variable logic has interesting connections to the foundations of XML. For instance, Core XPath 1.0 corresponds exactly to two-variable logic on unranked, ordered trees [12] and the regular tree languages, which capture the structural part of existing schema languages, exactly correspond to existential monadic second order logic with two first-order variables (see, e.g., [4]). Initially, most logic-based research on XML focused on abstractions by labeled trees that only took the structure into account and ignored the data in XML documents.

[*]We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599

However, in recent years, data has attracted more attention. In [4] XML documents are modeled by *data trees*, trees whose nodes have a label from a finite alphabet Σ and a data value from an infinite domain dom. Further work on this subject can be found in [9, 11, 13]. Similarly, languages of data words, where each position has a label and a data value, have been studied, sometimes with an additional motivation by potential applications in automatic verification [7, 5, 3].

The main results of [4, 5] are that

- satisfiability of two-variable logic over data words is decidable even if formulas can use the linear order and the successor relations on positions [5], and that
- satisfiability of two-variable logic over data trees is decidable if only the parent-child relationship and the direct sibling relationship between the children nodes of a parent are available [4].

The former problem has unknown (but probably huge) complexity, the latter can be solved in 3-NEXPTIME.

Even though two-variable logic can express a lot of interesting properties of XML documents, their ability to express integrity constraints is limited. More precisely, they can express in general key, foreign key and inclusion constraints only if they are unary.

In this paper, we aim to shed some light on the problem to decide whether, for a given formula of two-variable logic and a set of (not necessarily unary) key constraints, there is an XML document that fulfills the formula and the constraints. However, as a start, we study it here only for data *words*.

Integrity constraints for XML have been studied before, for example in [1, 2]. One of the results[1] of [1] is that, it can be decided in NEXPTIME whether there is an XML document for a given regular tree language and a set of key and foreign key constraints. This work also established a close relationship of key constraints with linear (and prequadratic) equations that also shows up in our approach.

Contributions and organisation. We define our notion of key constraints[2] in Section 2. The keys that we allow are relatively flexible as they allow to specify sets of symbols at each position of a key and they allow the use of wildcards

[1]The article contains much stronger results, but this one is most relevant to us

[2]From a databases point of view the term "uniqueness constraints" would probably be better, but it is common to refer to our type of constraints as keys.

for positions to specify that the data value of these positions is considered irrelevant for the key.

Our main result, which is presented in Section 4 is that it is decidable whether, for a given formula of two-variable logic with successor relation ($\mathrm{FO}^2(\sim,+1)$) and a set of key constraints, there is a data word that fulfills the formula and the constraints.

As an intermediate step towards the proof we give in Section 3 a 2-NEXPTIME upper bound for the satisfiability problem for $\mathrm{FO}^2(\sim,+1)$ over data words without key constraints. The best previous upper bound was inferred from the data tree case: 3-NEXPTIME. However, the proof uses basically the same techniques as in [4, 5] and only makes use of the simpler setting in the string case. Another self-contained proof for this problem with a 4-NEXPTIME upper bound has been recently presented in [6].

It remains unclear whether the 2-NEXPTIME bound is optimal. The best known lower bound is NEXPTIME and can be inferred from [8]. However, we show in Section 5 that the satisfiability problem for $\mathrm{FO}^2(\sim,+1,+3)$ over data words is 2-NEXPTIME-hard.

Acknowledgement. The second author thanks Luc Segoufin for pointing him to this problem some time ago and for fruitful discussions on the subject that already headed in the right direction. He also thanks Fritz Eisenbrand for useful suggestions concerning the solution of systems of prequadratic equations.

2. DEFINITIONS

A *data word with symbols* over an alphabet Σ and a data domain dom is a finite, non-empty sequence of pairs $w = (\sigma_1, d_1)\cdots(\sigma_n, d_n)$ where each $\sigma_i \in \Sigma$ and $d_i \in \mathrm{dom}$. A *data word with propositions* over a finite set $\mathcal{P}$ of propositions and a data domain dom is a finite sequence $w = (P_1, d_1)\cdots(P_n, d_n)$ where each $P_i \subseteq \mathcal{P}$ and $d_i \in \mathrm{dom}$. For data words with symbols, we denote the *length* n of w by $|w|$. We call the string $\mathrm{str}(w) =_{\mathrm{def}} \sigma_1\cdots\sigma_n$ the *string projection* of w, likewise for data words with propositions. The set of data values occurring in w is denoted by $\mathrm{dom}(w)$. For each $d \in \mathrm{dom}(w)$, the *class of* d *in* w is the set $\mathrm{Cl}_d(w)$ of positions with value d. A *zone* of a data word is a maximal substring in which all positions carry the same data value.

The *Parikh image* $\mathrm{Par}(w)$ of a Σ-word (or data word over Σ) is the function that maps every symbol in Σ to the number of its occurrences in w. The Parikh image $\mathrm{Par}(L)$ of a language is just the set $\{\mathrm{Par}(w) \mid w \in L\}$.

An *atomic* $\mathcal{P}$*-type* is a set of propositions and negated propositions from $\mathcal{P}$. The *full atomic* $\mathcal{P}$*-type* of a position i in a data word is the formula $\alpha_P(x) = \bigwedge_{p\in P_i} p(x) \wedge \bigwedge_{p\in\mathcal{P}-P_i} \neg p(x)$. The set of full atomic types over $\mathcal{P}$ is denoted by $\mathcal{T}(\mathcal{P})$. Clearly, there is a simple relationship between subsets $P \subseteq \mathcal{P}$ and full atomic $\mathcal{P}$-types α. Therefore we sometimes represent a full atomic type α by the set P of its positive propositions and can identify $\mathcal{T}(\mathcal{P})$ with $\mathrm{Pow}(\mathcal{P})$.

In Sections 3 and 5 we deal exclusively with data words over propositions. In Section 4, where we consider key constraints, we use words with symbols. Upper bounds for data words over propositions (as Theorem 3.4) translate to data words with symbols but not necessarily vice versa. Data words over a set $\mathcal{P}$ of propositions can be considered as data words with symbols over the alphabet $\mathcal{T}(\mathcal{P})$ (or $\mathrm{Pow}(\mathcal{P})$). Likewise, automata for data words with propositions can be defined as (usual) automata over $\mathcal{T}(\mathcal{P})$.

DEFINITION 2.1. A *key constraint* κ for words with symbols is a sequence of entries from $(\mathrm{Pow}(\Sigma) \times \{\bullet, \circ\})$. For a key constraint $\kappa = (K_1, \otimes_1)\cdots(K_k, \otimes_k)$ we call k the

We say that a key constraint $\kappa = (K_1, \otimes_1)\cdots(K_l, \otimes_l)$ *matches a position* i in a data word $w = (a_1, d_1)\cdots(a_n, d_n)$ over an alphabet Σ if for every $j \in [1, k]$, $a_{i+j-1} \in K_j$.

A key constraint is *violated* in w if it matches two different positions $i_1 \neq i_2$ and, for every $j \in \{1, \ldots, l\}$, if $\otimes_j = \bullet$ then $d_{i_1+j-1} = d_{i_2+j-1}$. Otherwise it is *fulfilled*. We write $w \models \kappa$ if κ is fulfilled in w and, for a set $\mathcal{K}$ of key constraints, $w \models \mathcal{K}$ if w fulfills every key constraint in $\mathcal{K}$. For data words with propositions the definition is analogous with $\mathcal{T}(\mathcal{P})$ in place of Σ and full types α_i in place of symbols a_i.

For a set $\mathcal{K}$ we denote the maximum length of a key in $\mathcal{K}$ by $k(\mathcal{K})$.

In this paper, a data word $w = (P_1, d_1)\cdots(P_n, d_n)$ with propositions from $\mathcal{P}$ is represented by a logical structure with universe $\{1, \ldots, n\}$, a successor relation $+1$, an equivalence relation $\sim$ that holds for two positions if they carry the same data value, and one unary relation p for every $p \in P$.

The logic $\mathrm{FO}^2(\sim,+1)$ is just first-order logic over such structures, restricted to the use of variables x and y. Thus, quantifiers range over positions of a data word, the formula $x+1 = y$ expresses that y is the right neighbor of x and $p(x)$ indicates that at position x proposition p holds. A formula $x \sim y$ expresses that at x and y the same data value occurs.

$\mathrm{FO}^2(\sim,+1)$ over words with symbols is defined analogously with atomic formulas $\sigma(x)$ for symbols $\sigma \in \Sigma$.

We denote consecutive sets of natural numbers by interval notation, for example $[3, 5] = \{3, 4, 5\} = (2, 5]$.

3. $\mathrm{FO}^2(\sim,+1)$ WITHOUT KEY CONSTRAINTS

In this section we deal exclusively with data words with propositions. An FO^2-formula is in *Scott normal form (SNF)* if it is of the form

$$\psi = \big(\forall x \forall y\ \chi \wedge \bigwedge_i \forall x \exists y\ \chi_i\big),$$

where χ and each χ_i are quantifier-free $\mathrm{FO}^2(\sim,+1)$ formulas (see [10] for a reference). In a standard fashion, any $\mathrm{FO}^2(\sim,+1)$ formula can be translated[3] into a formula in Scott normal form that is equivalent with respect to satisfiability, as stated in the following lemma.

LEMMA 3.1. *For each $FO^2(\sim,+1)$ formula φ a $FO^2(\sim,+1)$ formula φ' in Scott normal form can be computed in polynomial time such that φ is satisfiable over data words if and only if φ' is satisfiable over data words.*

PROOF. From φ one can compute an existential second-order formula

$$\psi = \exists R_1 \cdots R_m \big(\forall x \forall y\ \chi \wedge \bigwedge_i \forall x \exists y\ \chi_i\big),$$

that is equivalent to φ and where the relation symbols R_i are unary. Let $\mathcal{P}' = \mathcal{P} \cup \{p_1, \ldots, p_m\}$, where the propositions p_i are new. Let φ' be the formula obtained from ψ by

[3]If stated for data words with symbols this lemma would come with an exponential blow-up of the number of symbols due to the need to encode all possible combinations of the R_i relations.

removing the quantification of the relations R_j and replacing each atom $R_j(x)$ by $p_j(x)$ (and likewise each $R_j(y)$ by $p_j(y)$). Clearly, φ' is satisfied by some data word over $\mathcal{P}'$ if and only if φ is satisfied by some data word over $\mathcal{P}$. $\square$

Thus, we can assume henceforth that the $\mathrm{FO}^2(\sim,+1)$-formula φ that shall be tested for satisfiability is in Scott normal form.

In the following, we will annotate word positions by propositions that reflect the propositions of the adjacent positions and whether the own data value equals the data values of the adjacent positions. To this end we use additional propositions of the form p^{-1} and p^{+1}, for every $p \in \mathcal{P}$. For a subset $P \subseteq \mathcal{P}$, we define $P^{-1} = \{p^{-1} \mid p \in P\}$ and $P^{+1} = \{p^{+1} \mid p \in P\}$. Furthermore, we use the additional propositions $p_=^{-1}$ and $p_=^{+1}$ to indicate data equalities and $p_\rhd$ and $p_\lhd$ to mark border positions. Finally, the propositions p_1 and p_2 are used to mark up to two occurrences of a type in a class. By $\mathcal{P}_1$ we denote $\mathcal{P} \cup \mathcal{P}^{-1} \cup \mathcal{P}^{+1} \cup \{p_=^{-1} p_=^{+1}, p_\rhd, p_\lhd\}$ and by $\mathcal{P}_2 = \mathcal{P}_1 \cup \{p_1, p_2\}$. Clearly, $|\mathcal{P}_2| = \mathcal{O}(|\mathcal{P}|)$.

A data word $w' = (P'_1, d_1) \cdots (P'_n, d_n)$ over $\mathcal{P}_2$ is *valid* if it fulfills the following conditions.

(i) P'_1 contains $p_\rhd$ but not $p_=^{-1}$ and no proposition of the form p^{-1}.

(ii) P'_n contains $p_\lhd$ but not $p_=^{+1}$ and no proposition of the form p^{+1}.

(iii) If $i < n$ then P'_i contains a proposition p^{+1} if and only if P'_{i+1} contains p. Furthermore, it contains $p_=^{+1}$ if and only if $d_i = d_{i+1}$.

(iv) If $i > 1$ then P'_i contains a proposition p^{-1} if and only if P'_{i-1} contains p. Furthermore, it contains $p_=^{-1}$ if and only if $d_i = d_{i-1}$.

(v) If a class contains at least one position with a $\mathcal{P}_1$-type α then it contains exactly one such position with proposition p_1 (and at this position p_2 does not hold).

(vi) If a class contains at least two positions with a $\mathcal{P}_1$-type α then it contains exactly one such position with proposition p_2 (and at this position p_1 does not hold).

We call a data word over $\mathcal{P}_1$ valid if it fulfills all but the last two conditions. By $\mathrm{de}(w')$ we denote the data word over $\mathcal{P}$ that is obtained from a data word w' over $\mathcal{P}_2$ by dropping all other propositions. Clearly, for every data word w over $\mathcal{P}$ there is a unique valid data word w' over $\mathcal{P}_1$ with $\mathrm{de}(w') = w$. But there can be more than one such w' over $\mathcal{P}_2$.

In the following, we define class type functions for $\mathcal{P}_2$-words. The intention is that the class type function of a valid $\mathcal{P}_2$-word w' contains all relevant information to decide whether $\mathrm{de}(w') \models \varphi$.

If w' is a valid data word over $\mathcal{P}_2$ and c a class of w', the *class type* τ of c is the pair (S, D), where D is the set of all full types α occurring in c that contain p_1 or p_2 and S is the set of all full types α in c that do not contain p_1 or p_2. We call the types in D *dog types* of τ and the types in S *sheep types*. Note that if w' is valid then each class type of every class in w' fulfills that

- if it contains a $\mathcal{P}_1$-type α at all then D contains $\alpha \cup \{p_1, \neg p_2\}$, and
- if it contains a $\mathcal{P}_1$-type α in S then D contains $\alpha \cup \{\neg p_1, p_2\}$.

Furthermore, no full types containing $\{p_1, p_2\}$ occur in a class type. We call a class type *valid* if it fulfills these conditions.

By $\mathrm{CT}(\mathcal{P}_2)$ we denote the set of all valid class types. Clearly, $|\mathrm{CT}(\mathcal{P}_2)| \leq 2^{2^{\mathcal{O}(|\mathcal{P}|)}}$.

The *class type function* $\mathrm{ctf}_{w'}$ of a data word w' over $\mathcal{P}_2$ maps every class type to the number of classes of w' with this class type.

An ℓ-profile is a function $T : \mathrm{CT}(\mathcal{P}_2) \to [0, \ell) \cup \{*\}$. A class type (S, D) *occurs* in T if $T(S, D) \neq 0$. We say that a data word w' is a *solution* for an ℓ-profile T (short: $w' \models T$) if, for every class type (S, D) either $T(S, D) = \mathrm{ctf}_{w'}(S, D)$ or $T(S, D) = *$ and $\mathrm{ctf}_{w'}(S, D) \geq \ell$. An ℓ-profile that has a solution is called *satisfiable*.

We show in the following proposition that ℓ-profiles contain all necessary information to decide whether a data word satisfies an $\mathrm{FO}^2(\sim,+1)$ formula. More precisely, we show that either for all solutions w' of an ℓ-profile T it holds $\mathrm{de}(w') \models \varphi$ or for none. In the former case we call T *φ-compatible.*

PROPOSITION 3.2. *(a) Let $\ell \geq 2$ and let φ be a $FO^2(\sim,+1)$-formula in SNF. For each ℓ-profile T either for all solutions w' to T it holds $de(w') \models \varphi$ or for all solutions w' to T it holds $de(w') \not\models \varphi$.*

(b) There is an algorithm that on input φ and a satisfiable ℓ-profile T with $\ell \geq 2$, decides whether T is φ-compatible in time that is exponential in φ and linear in $|T|$.

PROOF. We explain in the following how it can be inferred from T whether for a data word w' with $w' \models T$ it holds $\mathrm{de}(w') \models \varphi$ and conclude (a) and (b). The proof is similar to the proofs of Lemmas 8 and 9 in [5].

Let φ be a formula of the form

$$\big(\forall x \forall y\ \chi \wedge \bigwedge_i \forall x \exists y\ \chi_i\big).$$

We show first how it can be inferred from T whether $\forall x \forall y\ \chi$ holds. It is straightforward to bring χ into CNF, and to rewrite $\forall x \forall y\ \chi$ as a conjunction of (exponentially many) formulas of the form:

$$\psi = \forall x \forall y \big((\alpha(x)\ \wedge\ \beta(y)\ \wedge\ \delta(x,y)) \ \rightarrow\ \gamma(x,y) \big),$$

where α and β are full atomic $\mathcal{P}$-types, $\delta(x,y)$ is either $x{\sim}y$ or $x{\not\sim}y$, and $\gamma(x,y)$ is a disjunction of some formulas from $x = y-1$, $x = y$, $x = y+1$ and $x \notin [y-1, y+1]$. The latter expression is an abbreviation for the formula $x \neq y-1 \wedge x \neq y \wedge x \neq y+1$. We show how it can be determined from $\mathrm{ctf}_{w'}$ whether for such a formula ψ it holds $\mathrm{de}(w') \models \psi$.

Before we continue, let us clarify the relationship between full $\mathcal{P}$-types and full $\mathcal{P}_2$-types. Each $\mathcal{P}_1$-type basically consists of the $\mathcal{P}$-type of a position, the $\mathcal{P}$-types of its left and right neighbor and the information whether the left and right neighbor have the same data value as the position. Thus, we can view a $\mathcal{P}_1$-type as a tuple $(\alpha, \alpha^{-1}, \alpha^{+1}, p_=^{-1}, p_=^{+1})$ of three full $\mathcal{P}$-types and two atomic propositions. In $\mathcal{P}_2$-types, the propositions p_1 and p_2 additionally mark up to two occurrences of every $\mathcal{P}_1$-type α in a class.

We distinguish two main cases, that depend on whether $x \notin [y-1, y+1]$ occurs in $\gamma(x,y)$.

Let us assume first that $x \notin [y-1, y+1]$ is *not* a disjunct of $\gamma(x,y)$, that is $\gamma(x,y)$ is a disjunction of formulas from $x = y-1$, $x = y$ and $x = y+1$. If $\delta(x,y)$ is $x{\sim}y$, then ψ can only hold if α and β occur in the same class only at positions of distance ≤ 1. Whether this is true in $\mathrm{de}(w')$ can be inferred by inspecting all class types occurring in T. For instance, if for some class type (S, D) occurring in T, D contains a type $(\alpha, \alpha^{-1}, \alpha^{+1}, p_{=}^{-1}, p_{=}^{+1})$ where $\beta \notin \{\alpha, \alpha^{-1}, \alpha^{+1}\}$ and there is a type $(\beta, ...)$ in D then ψ fails to hold. By a tedious but simple case distinction it can be shown that whether ψ holds can indeed be inferred from T.

Likewise, if $\delta(x,y)$ is $x \not\sim y$, whether ψ holds can be inferred by inspecting all class types containing α and those containing β.

Let us now assume that $\gamma(x,y)$ is a disjunction of the formula $x \notin [y-1, y+1]$ and some of the formulas from $x = y-1$, $x = y$ and $x = y+1$. In this case occurrences of α and β at distances > 1 can never make ψ fail. Thus,for such formulas, it is sufficient to check all occurring $\mathcal{P}_1$-types in T and to test them for compatibility with ψ.

We now turn to formulas of the form $\forall x \exists y\ \chi_i$.

The formula χ_i can be transformed into a disjunction of (possibly exponentially many) formulas of the form

$$\bigvee \big(\alpha(x) \wedge \beta(y) \wedge \delta(x,y) \wedge \gamma(x,y)\big),$$

where α, β, δ are as before and γ is just one of $x = y-1$, $x = y$, $x = y+1$ and $x \notin [y-1, y+1]$.

Instead of considering all possible cases, we illustrate the technique by a typical example. Let us assume that, for some α, the only three disjuncts involving α are

- $\theta_1 = \alpha(x) \wedge \beta_1(y) \wedge x \sim y \wedge x \notin [y-1, y+1]$,
- $\theta_2 = \alpha(x) \wedge \beta_2(y) \wedge x \not\sim y \wedge x \notin [y-1, y+1]$, and
- $\theta_3 = \alpha(x) \wedge \beta_3(y) \wedge x \not\sim y \wedge x = y-1$.

To decide whether every position with type α is consistent with χ_i, one can inspect all occurrences of α in the class types of T. An occurrence $(\alpha, \alpha^{-1}, \alpha^{+1}, p_{=}^{-1}, p_{=}^{+1})$ fulfills θ_3 only if $\alpha^{+1} = \beta$. For all other occurrences of α it has to be tested whether they fulfill θ_1 or θ_2. For such occurrences, there must either be a distant occurrence of β_1 in the same class or there must be a distant occurrence of β_2 in some other class. The former can be checked by inspecting the class type (and if $\alpha = \beta$ it is important that the class type indicates whether α occurs at least twice in a class). In the latter case, the occurrence of β_2 in some other class is guaranteed if β_2 occurs in some other class type or if it occurs in the same class type which, by the class type function, has at least two classes. The latter kind of test is the reason why we require $\ell \geq 2$ in the statement of the proposition.

Statement (a) now follows immediately, as $\mathrm{de}(w') \models \varphi$ implies that all tests on T go through, whereas if $\mathrm{de}(w') \not\models \varphi$ some test fails.

The algorithm for (b) computes the CNF for $\forall x \forall y\ \chi$ and the DNF for $\bigwedge_i \forall x \exists y\ \chi_i$, each of exponential size in $|\varphi|$. For each subformula, it inspects all class types in T and performs the tests described above. If all tests pass, it accepts. □

Proposition 3.2 almost yields a decision algorithm for $\mathrm{FO}^2(\sim, +1)$. This algorithm could guess a ℓ-profile T and test whether it is φ-compatible. However, it could happen that T does not have a solution. In that case, Proposition 3.2 does not guarantee a correct answer. Thus, we need an additional algorithm that tests satisfiability of ℓ-profiles. Then, we can decide satisfiability of φ by guessing T and testing that it is satisfiable and φ-compatible.

Our approach is an adaptation of the techniques of [4, Proposition 3.30] from trees to strings. The idea is to translate a given ℓ-profile T into conditions on the string projections of solutions to T. Then we show how it can be decided whether there is a string that satisfies these conditions and that from that string a solution to T can be inferred.

More precisely, we will construct a string automaton A (that actually only depends on $\mathcal{P}$) and a linear set Lin that basically contains a kind of Parikh images of string projections that are not forbidden by T.

However, in order to ensure that $L(A) \cap L(\mathrm{Lin}) \neq \emptyset$ implies that T has a solution, we need to add more detailed information to the strings we consider.

Let w' be a valid data word over $\mathcal{P}_1$. Recall that a zone of w' is a maximal substring in which all positions carry the same data value. Let c be a class in w' and let $\tau = (S, D)$ be its class type.

For each zone z of c, the *zone type* D_z of z is the set of dog types that occur in z. A zone z is called a *dog zone* if $D_z \neq \emptyset$ otherwise a *sheep zone*. Clearly, all sets D_z of a class are pairwise disjoint and together they contain all types from D, that is, they induce a partition of D.

We note also that each class c can have at most $2^{|\mathcal{P}_2|}$ dog zones.

The *partitioned class type* of c is the pair $(S, \{D_z\}_z)$. The number of different partitioned class types is bounded by an exponential in $\mathcal{P}_1$ (and thus in $\mathcal{P}$). We denote the set of all partitioned class types by $\mathrm{PCT}(\mathcal{P}_2)$. The *partitioned class type function* $\mathrm{pctf}_{w'}$ of w' is the function that maps every partitioned class type to its number of occurrences in w'. An *partitioned ℓ-profile* is a function $T' : \mathrm{PCT}(\mathcal{P}_2) \to [0, \ell) \cup \{*\}$, analogous to an ℓ-profile. The terms *solution* and *satisfiable* are defined analogous to ℓ-profiles.

There is a straightforward correspondence between partitioned class types and class types. If $(S, \{D_z\}_z)$ is the partitioned class type of a class then its class type is $(S, \bigcup_z D_z)$. Thus, partitioned profiles are just refinements of profiles and an ℓ-profile T is satisfiable if and only if one of its refinement partitioned ℓ-profiles is satisfiable.

In the following, we describe how to test whether a partitioned ℓ-profile (with large enough ℓ) is satisfiable.

To be able to use automata for (part of) this test, we first define, for every data word w', strings (without data) that encode informations about the zones and class types of w'. To this end, let for each partitioned class type τ, up to ℓ classes of w' of type τ be numbered $1, 2, .., \ell$. By $N(c)$ we denote the number of class c. If there are more than ℓ classes of a type we set, for the remaining classes c, $N(c) = *$.

Let $\Gamma' = \mathrm{PCT}(\mathcal{P}_2) \times \mathrm{Pow}(\mathrm{Pow}(\mathcal{P}_2)) \times ([1, \ell) \cup \{*\}) \times \mathrm{Pow}(\mathcal{P}_2)$ and $\Gamma = \Gamma' \cup \mathrm{Pow}(\mathcal{P}_2)$. With a data word $w' = (P_1, d_1) \cdots (P_n, d_n)$ over $\mathcal{P}_2$ with a numbering N, we associate a Γ-string $\mathrm{estr}_\ell(w', N) = v_1 \cdots v_n$ as follows. If position i is *not* the first position of a zone then $v_i = P_i$. If it is the first position of a zone z of a class c then $v_i = (\tau_c, D_z, N(c), P_i)$.

In $\mathrm{estr}_\ell(w', N)$ the zones of w' are just the substrings induced by position intervals $[i, j)$ where i is a Γ'-position and

$j > i$ is the next Γ'-position. We call each such substring in a Γ-string a *pseudo-zone*.

The automaton A accepts a Γ-string v if the following conditions[4] hold.

(i) v is valid with respect to the propositions in $\mathcal{P}_2$ (analogous to valid data words).

(ii) Each pseudo-zone z in v is consistent with respect to the profile information in its first symbol. More precisely, if (τ, D, j, α) is the first symbol of a pseudo-zone z then each dog type $\alpha \in D$ has to occur exactly once in z. Besides that only sheep types of τ occur (arbitrarily often) in z.

(iii) For every pair of adjacent pseudo-zones with initial symbols (τ, D, i, P) and (τ, D, i', P') either $i \neq i'$ or at least one of i and i' is $*$.

(iv) The pseudo-zones are consistent with respect to the propositions $p_=^{-1}$ and $p_=^{+1}$.

It is obvious that A only depends on $\mathcal{P}$ but not on the particular T'.

We next describe the linear constraints that we derive from a partitioned profile T'. Intuitively, a set $\text{Lin}_\ell(T')$ expresses that for each (τ, D, i), where D occurs in the partition of τ and $T'(\tau) \geq i$ there is exactly one zone in a word. Furthermore, for each τ, all zones of the type $(\tau, D, *)$ occur equally often.

In the following we consider functions from I to $\mathbb{N}$, for various sets I. Each such function can be considered as a vector with $|I|$ entries from $\mathbb{N}$. A *linear set* over I is a set of functions $I \to \mathbb{N}$ that can be represented as $\{f + \sum_i j_i f_i \mid j_i \in \mathbb{N}\}$, where f and all f_i are functions $I \to \mathbb{N}$.

For a Γ-string v, let h_v denote the function that maps every triple (τ, D, i) consisting of a partitioned class type τ, a dog set D and $i \in [1, \ell) \cup \{*\}$ to the number of symbols of the form (τ, D, i, P) in $\text{estr}(w')$.

$\text{Lin}_\ell(T')$ is the linear set that captures all functions h_v for strings of the form $\text{estr}_\ell(w', N)$ with $w' \models T'$.

For each partitioned class type τ let $e_{\tau,j}$, be the indicator function for the set $\{(\tau, D, i) \mid i \leq j\}$, that is it maps every triple from this set to 1 and all other triples to 0. Let $e_{\tau,*}$, be the indicator function for $\{(\tau, D, *)\}$ to 1. Then $\text{Lin}_\ell(T')$ is simply the set of all functions of the form

$$\sum_{\tau, T'(\tau) \neq *} e_{\tau, T'(\tau)} + \sum_{\tau, T'(\tau) = *} e_{\tau,\ell} + i_\tau e_{\tau,*},$$

with arbitrary natural numbers i_τ.

Proposition 3.3. *Let $\ell = 2^{|\mathcal{P}_2|}$ and let T' be a partitioned ℓ-profile. If there is a Γ-string $v \in L(A)$ with $h_v \in$ $Lin_\ell(T')$ then there is a data word w' and a numbering N such that $estr_\ell(w', N) = v$ and $w' \models T'$.*

Proof. Let $v \in L(A)$ with $h_v \in \text{Lin}_\ell(T')$. We define the data word w' by assigning data values to the positions of v and replacing every symbol of the form (τ, D, i, P) by P. For each partitioned type τ we pick[5] a pseudo-zone dog type D and set $o_v(\tau) = \sum_i h_v(\tau, D, i)$. Thus, $o_v(\tau)$ is the number of classes of type τ in v. We reserve an ordered set Δ_τ of $o_v(\tau)$ data values for each partitioned type τ. We call a pseudo-zone *special* if its first symbol is of the form (τ, g, i, P) with $i \neq *$.

We assign the data values in three stages as follows:

(1) to all special pseudo-zones,

(2) to all non-special dog pseudo-zones,

(3) to all non-special sheep pseudo-zones.

Assigning data values to special pseudo-zones is very simple. If z is a pseudo-zone with initial symbol (τ, g, i, P) we assign the i-th data value of Δ_τ to z. Note that $\text{Lin}_\ell(T')$ makes sure that, for every τ, every pseudo-zone dog type D of τ and every $i \leq \min(\ell, h_v(\tau))$, there is exactly one symbol (τ, D, i, P) in v. After stage (1) only data values need to be assigned for classes whose profile occurs more than ℓ times and for each of those, the ℓ special classes already received a data value.

We assign data values to non-special dog pseudo-zones inductively from left to right. To this end, let z be a pseudo-zone of type D for some type τ, and let d be the data value assigned to z's left neighbor (undefined, if it is not yet assigned). If there is a data value different from d in Δ_τ that has not yet been assigned to a (τ, D)-pseudo-zone, then this value can be chosen for z. However, if d is the only value in Δ_τ that has not yet been assigned we will assign d to some pseudo-zone z' with profile D which already has a value $d' \neq d$ and use d' for z. However, we have to make sure that none of the neighbor pseudo-zones of z' has the value d. As each class has at most $2^{|\mathcal{P}_2|}$ dog zones altogether, d can be assigned at this point to at most $2^{|\mathcal{P}_2|} - 1$ (dog) pseudo-zones each of which can be a neighbor to at most 2 other pseudo-zones. As there are at least $\ell - 1$ candidate pseudo-zones to choose from, it follows from the choice of ℓ that one of the candidates does not have a d-valued neighbor.

Finally, it is easy to assign values to non-special sheep pseudo-zones z for a type τ as z has only two neighbor pseudo-zones and $|\Delta_\tau| > \ell > 2$.

The numbering N can be inferred from the orders of the sets Δ_τ.

This completes the proof of the proposition. $\square$

The following theorem is formulated for data words with propositions but it also holds for data words with symbols.

Theorem 3.4. *The satisfiability problem for $FO^2(\sim, +1)$ formulas over data words with propositions is decidable in* 2-nexptime.

Proof. Let $\ell = 2^{|\mathcal{P}_2|}$ and let φ be the SNF of the input $\text{FO}^2(\sim, +1)$-formula. The non-deterministic algorithm first guesses an ℓ-profile T and a partitioned ℓ-profile T' that refines T and tests, using A and $\text{Lin}_\ell(T')$ if there is a Γ-string $v \in \text{Lin}_A$ with $h(v) \in \text{Lin}_\ell(T')$. It rejects, if this test fails. Otherwise, it runs the algorithm of Proposition 3.2 and accepts if it accepts.

Clearly, if there is a data string w with $w \models \varphi$ there is a φ-compatible ℓ-profile T with a solution w' such that $\text{de}(w') \models \varphi$ and a satisfiable partitioned ℓ-profile T' that refines T. Furthermore, $\text{estr}_\ell(w', N)$ is accepted by A (for any numbering N) and $h_{\text{estr}_\ell(w',N)} \in \text{Lin}_\ell(T')$, thus the algorithm is complete.

[4]We note that A does not need to test that D occurs in τ. This is implicitly handled by $\text{Lin}_\ell(T')$ below.

[5]Note that by construction we always have $D \neq \emptyset$ and that $\text{Lin}_\ell(T')$ guarantees that all pseudo zones occur equally often. Thus, the very choice of D does not matter.

To show soundness let T, T' be as above and let $v \in L(A)$ with $h_v \in \mathrm{Lin}_\ell(T')$. By Proposition 3.3 there is a data word w' and a numbering N with $\mathrm{estr}_\ell(w', N) = v$ and $w' \models T'$. Thus, T' and therefore T has a solution. Finally, as the algorithm of Proposition 3.2 correctly decides φ-compatibility for satisfiable profiles we can conclude that T is φ-compatible. and thus φ is indeed satisfiable.

It remains to show that the algorithm works in doubly exponential time. As $|\mathcal{P}|$ and $|\mathcal{P}'|$ are linear in $|\varphi|$, $|\mathrm{Pow}(\mathcal{P}_1)|$ is at most exponential in $|\varphi|$. Therefore, T and T' are objects of at most doubly exponential size. $\mathrm{Lin}_\ell(T')$ has functions with a doubly exponential domain and coefficients of at most doubly exponential size. Likewise, A has at most doubly exponential many states. Indeed, A only has to check local consistency of pseudo zones which basically requires states of the form (α, D), where α is the $\mathcal{P}_2$-type of the last symbol and D indicates which types need to be met in the current pseudo-zone.

Therefore, (see, e.g., [14, Proposition 4.3]) the Parikh image of $L(A)$ has coefficients of at most doubly exponential size. Thus, it can be tested non-deterministically in doubly exponential time (in $|\varphi|$) whether there is a string $v \in L(A)$ with $h_v \in \mathrm{Lin}_\ell(T')$. $\square$

It is open whether this bound is optimal. However, we show in Section 5 that if we additionally allow the formula to test whether two positions have distance 3 the problem becomes indeed 2-NEXPTIME-hard.

4. $\mathrm{FO}^2(\sim, +1)$ WITH KEY CONSTRAINTS

This section is devoted to the proof of the main theorem of this paper.

THEOREM 4.1. *It is decidable whether for a given $FO^2(\sim, +1)$ formula φ and a set $\mathcal{K}$ of key constraints there is a data word w such that $w \models \varphi$ and $w \models \mathcal{K}$.*

The result holds for data words with symbols as well as for data words with propositions.

Before we give the details of the proof we first discuss its general strategy and the underlying ideas.

Just as in Section 3 we (non-deterministically) construct an automaton A from the given formula that tests some regular conditions and a set $\mathcal{F}$ of restrictions on the occurrences of class types. From A and $\mathcal{F}$ semi-linear sets can be derived that basically describe the Parikh images of (simultaneous) solutions of A and $\mathcal{F}$. But this time we are interested in solutions that additionally fulfill the given key constraints.

Even though the basic idea of the construction is as in Section 3, there are some crucial differences.

- The precise shape of the underlying alphabet and the exact conditions are different,
- we (show that we can) restrict attention to data words
  - in which all zones have length one, and
  - each symbol occurs in (classes of) only one class type and thus the class type can be inferred from a symbol.

It turns out that key constraints of different "arities" (number of $\bullet$-entries) can be handled by different strategies. If a key has arity zero it basically states that certain string patterns should not occur more than once. This can be checked by a finite automaton. If a key has arity one then it basically states that in the same class the set of symbols of the $\bullet$-position can occur at most once within the context given by the key. This can be translated into conditions on the occurring class types.

Thus, it mainly remains to deal with key constraints of arity ≥ 2. To illustrate the idea let us consider a simple key of the form $(\{a\}, \bullet), (\{b\}, \bullet)$. If one of the symbols a or b is a dog in its class type, that is, it is allowed to occur only once in each class, then the key can not be violated (if the class type restrictions are met). Thus, we can assume that a and b are sheep[6]. If we know that the class types of a and b occur frequently in the solutions of A and $\mathcal{F}$ that we consider, say $\omega(n)$ many times where n is the length of the data word, then we have a lot of freedom to assign data values to occurrences of a and b. More precisely, there is a quadratic number of possible assignments of pairs of data values to substrings ab but, of course, there can be only a linear number of occurrences of such substrings in any data string. Thus, it will turn out that in this case the key constraint can be fulfilled.

The general strategy for the decision algorithm can be stated as follows. It computes a linear set describing the "Parikh image" of a set of solutions of A and $\mathcal{F}$. If the Parikh image tells that some class type occurs only a bounded number of times in such solutions, the symbols of this class are replaced by "special" symbols that already encode the data values for the bounded number of classes. A, $\mathcal{F}$ and $\mathcal{K}$ are adapted accordingly. This kind of operation might remove bullets from keys (as the data values are already encoded in special symbols, occurrences of these symbols do not need a $\bullet$). Therefore some keys might become nullary or unary, thus they can be replaced by new regular and class type conditions. This process might repeat until either a linear set is met in which all class types are bounded (which can then be tested by exhaustive search) or a set in which all class types are unbounded. In the latter case, the above observation about keys of arity ≥ 2 then guarantees the existence of a solution.

We note that as we anyway can not give an elementary upper bound for the overall algorithm we do not always aim to represent conditions on solutions in the most efficient way. Instead we favor simplicity over algorithmic efficiency.

The proof consists of three mains parts. First we show in Subsection 4.1 how the problem of satisfiability of $(\varphi, \mathcal{K})$ over data words can be (non-deterministically) reduced to the question whether an instance of a certain constraint problem over data words has a solution. Next, it is shown in Subsection 4.2 that if all class types in a linear set of solutions to A and $\mathcal{F}$ are unbounded and there are only keys of arity ≥ 2 then there is a solution to the constraint instance. Finally, we show in Subsection 4.3 how such an instance can be obtained from an arbitrary instance.

4.1 Preparations

We call a data word *separated* if two neighbor positions always have different data values, that is, if all zones are of size one. The constraint problem that we consider next is defined over separated data words with symbols. A *constraint instance* I is a tuple $(\mathcal{A}, \mathcal{F}, \mathcal{C}, \mathcal{K})$ consisting of

[6]However, in general it is not *that* simple due to the fact that keys have sets of symbols at a position.

- a string automaton $\mathcal{A}$,
- a family profile $\mathcal{F}$ (to be defined below),
- a set $\mathcal{C}$ of class types with non-empty dog sets, and
- a set $\mathcal{K}$ of key constraints.

We consider constraint instance for data words with symbols. All conditions refer to the alphabet Σ of the automaton $\mathcal{A}$.

A *class type family* (*family* for short) is a set $(S, \mathcal{D})$, where $S \subseteq \Sigma$ and $\mathcal{D} = \{D_1, \dots, D_m\}$, for some n and every $D_i \subseteq \Sigma$ is non-empty. All sets of a family are pairwise disjoint. A *family profile* is a set of families with pairwise disjoint sets of symbols. Each family $(S, \{D_1, \dots, D_m\})$ gives rise to m class types (S, D_i). We call the class types of $\mathcal{C}$ and the symbols therein *special*.

We further require[7] that $\mathcal{A}$ tests that for each class type (S, D) of $\mathcal{C}$, each symbol of D occurs exactly once in the string and otherwise symbols of this type are only from S.

A data word w is a *solution* to I ($w \models I$) if the following conditions hold.

- w is separated.
- The string projection of w is accepted by $\mathcal{A}$.
- Every class has a class type of a family from $\mathcal{F}$ or a class type from $\mathcal{C}$.
- $w \models \mathcal{K}$.

We call I *satisfiable* if it has a solution.

The following proposition states that the satisfiability problem for $\mathrm{FO}^2(\sim, +1)$ in the presence of key constraints can be reduced to satisfiability of constraint instances.

Proposition 4.2. *There is a non-deterministic algorithm that computes, for each pair $(\varphi, \mathcal{K}')$, where φ is a $FO^2(\sim, +1)$ formula and $\mathcal{K}'$ is a set of key constraints, a constraint instance $I = (\mathcal{A}, \mathcal{F}, \mathcal{C}, \mathcal{K})$ such that*

$(\varphi, \mathcal{K}')$ is satisfiable if and only if some computation of the algorithm yields a satisfiable I.

Proof. The proof is basically the same for data words with propositions as for data words with symbols. Our exposition assumes data words with propositions. Let φ and $\mathcal{K}'$ be given. It should be noted that we do not require that solutions to $(\varphi, \mathcal{K})$ are separated. However, the seventh step of the algorithm will lead to a constraint problem over separated data words.

We can assume that φ is in SNF. The algorithm works in seven stages which are described in the following. At the end of each stage there is a new set of conditions and (a) and (b) holds for the set of possible computations. The objects that represent these conditions (alphabets, automata, profiles, keys) are numbered in accordance with the respective stage (Σ_i, $\mathcal{A}_i$, T_i, $\mathcal{K}_i$). If nothing else is stated, objects are inherited from the previous stage.

1. From formula to automaton/profile. The algorithm first guesses a 2-profile T_1 and tests that it is φ-compatible. By the proof of Proposition 3.2 it holds that for every data word w with $w \models \varphi$ and every w' with $\mathrm{de}(w') = w$ there is some φ-compatible T_1 with $w' \models T_1$. Thus, $(\varphi, \mathcal{K})$ has a solution if and only if some run of the algorithm guesses a φ-compatible T_1 with a solution w' for which $\mathrm{de}(w') \models \mathcal{K}'$. It should be noted that the construction in the proof of Proposition 3.2 guarantees that all dog sets are non-empty. This condition will be kept valid by all transformations described below.

Let $\mathcal{A}_1$ be the automaton that tests for the string projection of a data word over $\mathcal{P}_2$ whether it fulfills conditions (i), (ii), (v) and (vi) of the definition of validity and additionally for every $i > 1$ that a set P_i contains $p_=^{-1}$ if and only if P_{i-1} contains $p_=^{+1}$ and likewise for equality to the right.

Next, the algorithm computes a new set $\mathcal{K}_1$ of key constraints as follows. To this end, for every key constraint $\kappa' = (K_1', \otimes_1) \cdots (K_k', \otimes_k)$ from $\mathcal{K}'$ a new key constraint $\kappa = (K_1, \otimes_1) \cdots (K_k, \otimes_k)$ is defined by letting each K_i be the set of all full $\mathcal{P}_2$-types α whose restriction to $\mathcal{P}$ is in K_i'. We let $\mathcal{K}_1$ be the set of all these key constraints. Clearly, for every valid data word over $\mathcal{P}_2$ it holds $w' \models \mathcal{K}_1$ if and only if $\mathrm{de}(w') \models \mathcal{K}'$.

2. Normalize keys wrt zone structure. In the following step, the algorithm changes the key constraints to make them more homogeneous. Recall that every $\mathcal{P}_2$-type α of a valid data word w' indicates whether the right (and the left) neighbor position has the same data value. If in some set K_i of a key κ, there is a full type α that implies that the left neighbor has the same data value and a full type β that implies that the left neighbor has a different data value then κ can not be violated by two matches of which one has α at position i and the other β.

Therefore, every key κ can be split with respect to the equality types without changing its semantics. That is, every K_i is partitioned into four sets $K_i^{==}$, $K_i^{\neq\neq}$, $K_i^{=\neq}$, $K_i^{\neq=}$ with the obvious semantics and from every possible of the 4^k combinations of these sets a new key is formed that replaces κ. Clearly, for the thus constructed set $\mathcal{K}_2$ it holds that for every valid data word w', $w' \models \mathcal{K}_2$ if and only if $w' \models \mathcal{K}_1$.

3. Use symbols enriched by class types. Next, the algorithm does the transition from words with propositions to words with symbols that are enriched by class types. Let Σ_2 be the set of all pairs (α, τ) where α is a full $\mathcal{P}_2$-type and τ a class type. For every valid data word $(\alpha_1, d_1) \cdots (\alpha_n, d_n)$ we define the Σ_2-word w' as $((\alpha_1, \tau_1), d_1) \cdots ((\alpha_n, \tau_n), d_n)$, where each τ_i is the class type of the class of position i. We let T_3 be the ℓ-profile resulting from T_2 by replacing every full type α in a class type τ by (α, τ). Correspondingly, $\mathcal{K}_3$ is constructed by replacing every α in a set K_i of a key κ by all pairs (α, τ) for which α is dog or sheep in τ. Clearly, for every valid $\mathcal{P}_2$-word w, $w \models T$ if and only if $w' \models T_3$ and $w \models \mathcal{K}_2$ if and only if $w' \models \mathcal{K}_3$.

Let $\mathcal{A}_3$ be the automaton that is obtained from $\mathcal{A}_1$ by allowing all symbols (α, τ) in place of α and does some further tests, described next.

4. Dissolve bounded class types Next, the algorithm singles out one or two special class types per class type in T_3. First, if $T_3(\tau) = 1$ for a class type τ, then this class type becomes special, that is, it is put into $\mathcal{C}_4$ but not into T_4. If $T_3(\tau) = *$ then for each symbol σ from τ two new symbols $(\sigma, \tau, 1)$ and $(\sigma, \tau, 2)$ are introduced and two new special class types with these symbols are put to $\mathcal{C}_4$. However, in this case, the "original" τ is put into T_4. The automaton $\mathcal{A}_4$ has to check that the occurrence of special symbols is

[7]This condition introduces some redundancy in I. However, it is convenient as $\mathcal{A}$ guarantees that special classes are handled correctly and $\mathcal{C}$ enables us to easily refer to special class types.

correct with respect to the special classe. In $\mathcal{K}_4$, the new symbols $(\sigma, \tau, 1)$ and $(\sigma, \tau, 2)$ are added to all sets K_i containing (σ, τ).

5. Normalize keys wrt class types. The next transformation step aims at making keys even more homogeneous. If a set K_i contains two symbols (α_1, τ_1) and (α_2, τ_2) with class types $\tau_1 \neq \tau_2$ then the key constraint can not be violated by two matches of κ, one involving (α_1, τ_1) and the other (α_2, τ_2). Therefore, the algorithm splits every set K_i of a key κ into sets K_i^τ with respect to their class type τ. The new keys obtained from κ are all combinations of these sets. If in this manner a key κ is created that only has special symbols then this key is removed and the responsibility to check it is delegated to the automaton $\mathcal{A}_5$. More precisely, $\mathcal{A}_5$ has to test that only one position in the string projection is matched by κ. Let $\mathcal{K}_5$ be the set of key constraints that are obtained in this way.

Clearly, for every valid data word w it holds $w \models \mathcal{K}_4$ if and only if $w \models \mathcal{K}_5$.

6. Dissolve unary keys. We say that a key $\kappa = (K_1, \otimes_1) \cdots (K_k, \otimes_k)$ is *single-valued* if all $\bullet$-positions are in one zone, that is, for all $i \in [j_1, j_2)$, where j_1 and j_2 are the first and last $\bullet$-postion in κ, respectively, $p_=^{\pm 1}$ occurs in the symbols of K_i. The following step replaces single-valued keys by class type constraints. To this end, we first define, for every single-valued key and each symbol σ in K_{j_1} a new symbol (σ, κ). Let $\mathcal{A}_6$ be the automaton obtained from $\mathcal{A}_5$ by allowing symbols (σ, κ) in place of σ which additionally tests that

- a symbol (σ, κ) only occurs at positions i if κ matches $i - j_1 + 1$, and
- symbols σ only occur at positions where none of the new symbols (σ, κ) matches.

Finally, class types (S, D) which contain symbols σ that were involved in the transformations just explained are adapted. More precisely, whenever a new symbol (σ, κ) is added the following is done, for the unique class type (S, D) in which σ occurs.

- If $\sigma \in S$ then a new class type $(S', D' \cup \{(\sigma, \kappa)\}$ is added, where S' results from S by picking a new symbol ρ', for every $\rho \in S$ (and likewise D' is obtained from D). $\mathcal{A}_5$ and $\mathcal{K}_5$ are changed accordingly to deal with the newly introduced symbols.
- If $\sigma \in D$ then a new class type $(S', (D' - \sigma) \cup \{(\sigma, \kappa)\}$ is added, where S' and D' are constructed as in the previous case.

7. Shrink zones. The last step of the algorithm replaces zones of length > 1 by singleton symbols as explained in the following. Thus, after this step, only separated solutions are sought.

The idea is to replace every zone z in a data word by a single position that contains all relevant information about z. Clearly, the data value of z can be just kept as data value for the position. Thus, it suffices to encode the necessary information about the symbols in z into a single symbol. More precisely, there are two kinds of new symbols:

- Symbols of the form $(\sigma_1, \ldots, \sigma_m)$, where $\sigma_i \in \Sigma_6$, $m \leq k(\mathcal{K}_6)$. Here, σ_1 must be the first symbol of a zone (indicated by $p_=^{-1}$ or $p_\triangleright$) and σ_m a last symbol of a zone. All other symbols have to indicate equal data equality with both neighbors. Furthermore, dogs are not allowed to occur more than once. These symbols are used to replace zones of lengths up to $k(\mathcal{K}_6)$.
- Symbols of the form (u, v, E, γ), where u and v are $(k(\mathcal{K}_6) - 1)$-tuples of symbols from Σ_6, $E \subseteq \Sigma_6$ and $\gamma : Q_5 \to Q_5$. These symbols are meant to replace zones z of length $> k(\mathcal{K}_6)$ containing exactly the symbols from E, for which u is the concatenation of the first (leftmost) $k(\mathcal{K}_6) - 1$ symbols, v is the concatenation of the last $k(\mathcal{K}_6) - 1$ symbols and γ is the transition function with respect to $\mathcal{A}_5$ that is induced by the string of z. We only allow symbols that are *consistent* in the sense that all symbols from u and v occur in E and there is a string w that starts with u, ends with v and uses exactly the symbols from E that induces[8] the transition function γ. Furthermore, the first symbol of u has to be a first symbol of a zone, the last symbol of v has to be a last symbol of a zone and all other symbols indicate equal data values in both neighbors.

The alphabet Σ_7 of A_7 only consists of these new symbols. Note that symbols σ that imply, that their data value is different from the data values of their neighbors are replaced by tuples (σ) of length 1.

Let $\mathcal{A}_7$, T_7 and $\mathcal{K}_7$ be the automaton, profile and key set obtained from the previous ones by the following adaptations.

- In $\mathcal{A}_7$, a symbol $(\sigma_1, \ldots, \sigma_m)$ has the same effect as the string $\sigma_1 \cdots \sigma_m$, a symbol of the form (u, v, E, γ) just has the effect of γ.
- Let $B(\sigma)$ be the set of symbols of the zone replaced by σ according to the above explained intention, that is $B((\sigma_1, \ldots, \sigma_m)) =_{\text{def}} \{\sigma_1, \ldots, \sigma_m\}$ and $B((u, v, E, \gamma)) =_{\text{def}} E$. Let (S, D) be a class type of T_6. Then we add the family $F = (S', \mathcal{D})$ to $\mathcal{F}$, with $S' = \{\sigma \in \Sigma_7 \mid B(\sigma) \subseteq S\}$. $\mathcal{D}$ contains each set $D' \subseteq \Sigma_7$, such that
  - $B(\sigma) \cap D \neq \emptyset$ for all $\sigma \in D'$,
  - $\bigcup_{\sigma \in D'} B(\sigma) \subseteq S \cup D$,
  - $\bigcup_{\sigma \in D'} B(\sigma) \cap D = D$, and
  - $D \cap B(\sigma_1) \cap B(\sigma_2) = \emptyset$ for all $\sigma_1, \sigma_2 \in D'$ with $\sigma_1 \neq \sigma_2$.

  The conditions are chosen such that only symbols corresponding to zones with a dog symbol are put into $\mathcal{D}$, and that each dog set D' contains each dog symbol from D exactly once.
- In keys with zones of length > 1 the respective sequences of sets are replaced by suitable sets of symbols of the form $(\sigma_1, \ldots, \sigma_m)$. If the zone occurs at one of the ends of the key, also symbols of the form (u, v, E, γ) have to be taken into account. Note that in this case, u or v contains all relevant information.

[8]It should be mentioned that there is an implicit control that dog symbols occur only once. Otherwise, the range of γ contains only rejecting sink states.

Finally, $I = (\mathcal{A}_7, \mathcal{F}, \mathcal{C}_7, \mathcal{K}_7)$ where the families of $\mathcal{F}$ are just $(S, \{D\})$, for every (S, D) class type (S, D) of T_7.

We omit the proof of the correctness of the algorithm as the correctness of each step should be clear. it should be noted that the only step that involves non-determinism is the first step that chooses a profile T. $\square$

Note that there can be an exponential blow-up in stage 7.

4.2 Fulfilling the Key Constraints

In this section we give a sufficient condition for the existence of a solution for a constraint instance $I = (\mathcal{A}, \mathcal{F}, \mathcal{C}, \mathcal{K})$.

The following lemma will be helpful as it allows us to focus on solutions w in which the number of classes is linear in the length of w.

Lemma 4.3. *For each solution of a partial constraint instance $I = (\mathcal{A}, \mathcal{F}, \mathcal{C})$ there is a solution w with the same frequencies of class types and $|w| \le |Q| \times (|\Sigma| \times |dom(w)| + 1)$.*

Proof. Let w be a solution for I of minimal length for a certain domain $\mathrm{dom}(w)$. Towards a contradiction, let us assume $|w| > |Q| \times (|\Sigma| \times |\mathrm{dom}(w)| + 1)$.

As each class contains at most $|\Sigma|$ dogs, w contains at most $|\Sigma| \times |\mathrm{dom}(w)|$ dog positions. Here, we count dog positions of special and non-special classes. These dog positions induce at most $|\Sigma| \times |\mathrm{dom}(w)| + 1$ maximal substrings of sheep. Thus, there must be a substring u of w that contains only sheep and has length at least $|Q| + 1$. Let us consider any accepting run of A. Just as in the proof of the pumping lemma there are two positions $i < j$ in u for which A assumes the same state after reading position i and j, respectively. Thus, the data word w' obtained from w by eliminating all positions in $(i, j]$ is also accepted by A. As these positions only contain sheep, w' also satisfies all constraints, thus it is a shorter solution, the desired contradiction. $\square$

For a finite automaton $\mathcal{A}$, $\mathrm{Par}(L(\mathcal{A}))$ is a *semi-linear set* over Σ, that is $\mathrm{Par}(L(\mathcal{A}))$ can be written as a union of linear sets over Σ. Furthermore, the linear sets can be represented in the form $\{f + \sum_i j_i f_i \mid j_i \in \mathbb{N}\}$, where the range of f only contains numbers $\le |Q|^2$ and the ranges of the f_i only contain numbers $\le |Q|$, where Q is the state set of $\mathcal{A}$ [14, Proposition 4.3].

With a family profile $\mathcal{F}$ and profile set $\mathcal{C}$ we associate a linear set $\mathrm{Lin}_{\mathcal{F},\mathcal{C}}$ as follows. For each dog set D occurring in $\mathcal{F} \cup \mathcal{C}$ let e_D be the indicator function for D. For each σ in a sheep set S let e_σ be the indicator function for $\{\sigma\}$. Then, $\mathrm{Lin}_{\mathcal{F}}$ is defined[9] as

$$\left\{ \sum_D i_D e_D + \sum_{\sigma \in S} i_\sigma e_\sigma \mid i_D, i_\sigma \in \mathbb{N} \right\}.$$

Clearly, if a data word w is a solution for $(\mathcal{A}, \mathcal{F}, \mathcal{C}, \mathcal{K})$ then $\mathrm{Par}(w) \in \mathrm{Par}(L(\mathcal{A})) \cap \mathrm{Lin}_{\mathcal{F},\mathcal{C}}$. We next give a sufficient condition for the reverse direction. Let $L_0 = \{f + \sum_j i_j f_j \mid i_j \in \mathbb{N}\}$ be a linear set such that $L_0 \subseteq \mathrm{Par}(L(\mathcal{A})) \cap \mathrm{Lin}_{\mathcal{F},\mathcal{C}}$. For a class type $\tau = (S, D)$ we let $f(\tau) =_{\mathrm{def}} f(\sigma)$ and $f_j(\tau) =_{\mathrm{def}} f_j(\sigma)$, for every[10] $\sigma \in D$. We call τ *bounded* with respect to L_0, if for all j it holds $f_j(\tau) = 0$. We call L_0 *unbounded* if all non-special class types are bounded with respect to L_0.

[9]The attentive reader might notice that this definition is overly "generous". However, it is sufficient for our purposes as the automaton controls that dogs from $\mathcal{C}$ only occur once.

[10]Note that this is independent of the choice of σ.

Proposition 4.4. *Let $I = (\mathcal{A}, \mathcal{F}, \mathcal{C}, \mathcal{K})$ be a constraint instance such that*

- *$\mathcal{A}$ controls that there are no key violations that involve matches with only special symbols and no special symbols of the same class occur next to each other,*
- *all key constraints in $\mathcal{K}$ are at least binary,*
- *there is no key constraint κ, which contains a special sheep symbol at a $\bullet$-position, and*
- *there is an unbounded linear set in $Par(L(\mathcal{A})) \cap Lin_{\mathcal{F},\mathcal{C}}$.*

Then I is satisfiable.

Proof. The proof is similar to the proof of Proposition 3.3. However, we have to take key constraints into account.

Let $I = (\mathcal{A}, \mathcal{F}, \mathcal{C}, \mathcal{K})$ be with the stated properties. Let Σ and Q be the alphabet and state set of $\mathcal{A}$, respectively.

Let $L_0 = \{f + \sum_j i_j f_j \mid i_j \in \mathbb{N}\}$ be an unbounded linear set in $\mathrm{Par}(L(\mathcal{A})) \cap \mathrm{Lin}_{\mathcal{F},\mathcal{C}}$. Let c' be the maximum value occurring in f or any of the f_j and let c be $c'(|\mathcal{F}| + |\mathcal{C}|)$. Let further $m = k(\mathcal{K})$ and $M = 40m^2c|Q| \times |\Sigma|$. Let g be the function obtained from L_0 by setting $i_j := M$, for all j, and let $w \in L(\mathcal{A})$ with $\mathrm{Par}(w) \in \mathrm{Lin}_{\mathcal{F},\mathcal{C}}$ be a string (without data) solution to $(\mathcal{A}, \mathcal{F}, \mathcal{C})$ of minimal length with $\mathrm{Par}(w) = g$.

Thus, the following statements hold.

- Every non-special class type in w occurs at least M times (by the choice of the i_j).
- $|\mathrm{dom}(w)| \le cM$ (as each class type occurs at most $c'M$ times).
- $|w| \le 2|Q| \times |\Sigma| \times |\mathrm{dom}(w)|$, as guaranteed by Proposition 4.3.

We have to show how data values can be assigned to the positions of w such that the resulting data word is separated and does not violate any key constraints.

We reserve, for each non-special class type $\tau = (S, D)$ a disjoint set Δ_τ of $g(\tau)$ data values. Here $g(\tau)$ is a shorthand for $g(\sigma)$, for some $\sigma \in D$, representing the number of τ-classes according to g. For a family $F = (S, \mathcal{D})$ we let $\Delta_F =_{\mathrm{def}} \cup_{D \in \mathcal{D}} \Delta_{(S,D)}$.

For each special class type one distinct data value is assigned. The condition on $\mathcal{A}$ ensures that after this assignment no key constraints are violated and no neighbor positions have the same data value.

Then, in two phases data values can be assigned

(1) to all non-special dog positions,

(2) to all non-special sheep positions.

We assign data values to non-special dog positions inductively from left to right. Let i be the first position with a dog symbol $\sigma \in D$ that has not yet received a data value. Let $\tau = (S, D)$ be the corresponding class type. Let $d \in \Delta_\tau$ be a *remaining data value*, that is, a value that has not yet been assigned to a σ-position. There are two possibilities why d could not be assigned to i: first, position $i - 1$ could already carry d, or some key constraint κ could be violated by assigning d to i. If there is some remaining $d \in \Delta_\tau$ to which none of these possibilities applies, we assign d to position i. Otherwise, we are going to exchange d with some data value $d' \in \Delta_\tau$ that has been assigned to some other σ-position.

Let $m = k(\mathcal{K})$. In the following, we denote the set of data values that occur at some position of distance $< m$ from a dog position with a data value e by $N_m(e)$. Note that, as each class has at most $|\Sigma|$ dogs, $|N_m(e)| < 2m|\Sigma|$, for every data value e.

Let us assume that assigning d to i would violate a key constraint κ of length k. Then there are positions i_1, i_2 such that for all (at least two) $\bullet$-positions j of κ, i_1+j-1 already received a data value and i_2+j-1 already received a data value, unless $i = i_2+j-1$. Let $i' = i_2+j'-1 \neq i$ for some (other) $\bullet$-position j' of κ and let $\hat{d}$ be the data value assigned to i'. Then i' is a dog position as the only sheep to which data values have already been assigned are special and special sheep do not occur at $\bullet$-positions. As the two matches need to have identical data values at $\bullet$-positions, we can conclude that $d \in N_m(d')$. Therefore, the $m-1$ left neighbors together rule out at most $(m-1)2m|\Sigma|+1 \leq 2m^2|\Sigma|$ data values[11] for position i.

As we assume that there is no valid value available, at least $|\Delta_\tau| - 2m^2|\Sigma|$ values have thus already been assigned to σ-positions. Let d be one of the remaining data values in Δ_τ and let j be a σ-position with a data value d'. Exchanging d and d' can be impossible because of one of the following reasons.

(i) $j-1$ or $j+1$ carries d,

(ii) $i-1$ carries d',

(iii) assigning d to j would yield a key violation involving position j, or

(iv) assigning d' to i would yield a key violation involving position i.

Condition (i) can apply to at most $2|\Sigma|$ many σ-positions. as d is only assigned to dog positions yet. Condition (ii) applies to at most one σ-position. As above, Condition (iii) only applies if some dog position (which matches a $\bullet$-position of some key) at distance $\leq m$ from j carries a data value from $N_m(d)$. This can be the case for at most $4m^2|\Sigma|$ many σ-positions. Condition (iv) only applies if, for some data value d'' at a dog position of distance $\leq m$ from i, $d' \in N_m(d'')$. This can be the case for at most $2m^2|\Sigma|$ σ-positions. Altogether there are at least $|\Delta_\tau| - 2m^2|\Sigma|$ candidate positions for a value exchange of which at most $(6m^2+2)|\Sigma|+1$ positions would lead to some violation. As $|\Delta_\tau| > (8m^2+2)|\Sigma|+1$, there is a position with which an exchange is possible. This completes the description of phase (1).

In phase (2) we assign data values in a left-to-right fashion to non-special sheep positions. We maintain as invariant that each data value is assigned to at most $X = 4c|Q| \times |\Sigma|$ positions, that are no special sheep positions. Let i be the first non-special sheep position from a family F to which we have not yet assigned a data value. There are three conditions under which we can not assign a data value $d \in \Delta_F$ to i.

(i) $i-1$ or $i+1$ carry d,

(ii) assigning d to i would yield a key violation involving position i, or

(iii) d has already been assigned to X positions.

Clearly, (i) rules out at most 2 values from Δ_F. A data value d can only fulfill condition (ii) if for some value d' occurring at distance $< m$ from i, d has an occurrence at distance $< m$ from some dog position of d' (if d is special) or some sheep or dog position of d' (if d is not special). This rules out at most $4m^2X$ data values. Finally, at most $\frac{|w|}{|X|}$ data values can be ruled out by condition (iii).

However, it holds

$$\begin{aligned} 2+4m^2X &= 2+16m^2c|Q| \times |\Sigma| < \frac{M}{2}, \\ \frac{|w|}{X} &\leq \frac{2cM|Q| \times |\Sigma|}{4c|Q| \times |\Sigma|} = \frac{M}{2}. \end{aligned}$$

Thus, fewer than M data values are ruled out. As $|\Delta_F| \geq M$, it is possible to find a data value $d \in \Delta_F$ that does not violate any constraints and maintains the invariant. $\square$

4.3 Separating frequent from infrequent class types

In this section, we use slightly extended notions of constraint families and constraint instances. An *extended constraint family* $F = (S, \mathcal{D}, \mathcal{E})$ consists of a set S of sheep symbols, a set $\mathcal{D}$ of dog sets (just as before) and a set $\mathcal{E}$ of special dog sets that have to occur in exactly one class. An *extended constraint system* is a tuple $(\mathcal{A}, \mathcal{F}, \mathcal{C}, \mathcal{K})$, where $\mathcal{A}, \mathcal{C}, \mathcal{K}$ are as before and $\mathcal{F}$ is an extended family.

There is a little subtlety with this notion. If a class has the class type of a set $E \in \mathcal{E}$ for some extended family F, then the dogs of this class are considered special symbols but the class could well contain non-special sheep symbols from S. If a key contains an entry K_i consisting of symbols from E and S, the intention is that it only disallows matches where symbols from S from the special class of E occur. Therefore, in the following we consider a key of this kind only violated if the i-positions of both matches of the violation carry either a symbol from E or a symbol from S from the class of E.

PROPOSITION 4.5. *There is a non-deterministic algorithm that for each extended constraint instance I with at least one family either accepts or computes an extended constraint instance I' such that the following conditions hold.*

(a) If the algorithm accepts then I is satisfiable.

(b) If the algorithm computes a satisfiable instance I' then I is satisfiable.

(c) If I is satisfiable then the algorithm has some computation that accepts or computes a satisfiable instance I' with fewer families than I.

PROOF. The algorithm uses many of the ideas and techniques that were already used in the proofs of Propositions 4.2 and 4.4.

Let $I = (\mathcal{A}, \mathcal{F}, \mathcal{C}, \mathcal{K})$ be the input constraint instance. The algorithm first guesses a linear set $L_0 = \{f + \sum_j i_j f_j \mid i_j \in \mathbb{N}\}$ and verifies that $L_0 \subseteq \text{Par}(L(\mathcal{A})) \cap \text{Lin}_{\mathcal{F}}$. If L_0 is unbounded the algorithm accepts.

[11]The $+1$ accounts for the fact that i needs a different data value than $i-1$.

If L_0 is not unbounded, there must be an extended family $F = (S, \mathcal{D}, \mathcal{E})$ for which some class type $\tau = (S, D)$ is bounded.

The algorithm continues by replacing τ by some new types in $\mathcal{E}$ in a way that is somewhat similar to step 4 of the algorithm in the proof of Proposition 4.2. For each symbol $\sigma \in D$, the algorithm introduces $f(\tau)$ new special symbols $(\sigma, 1), \ldots, (\sigma, f(\tau))$ and removes σ. It adds the $f(\tau)$ new resulting special class types to $\mathcal{E}$. These class types consist entirely of dogs. $\mathcal{A}$ is adapted in order to guarantee that no two symbols of the form (σ, j) and (σ', j) occur next to each other and to test that each (σ, j) occurs exactly once. Besides that, it handles (σ, j) just as it handled σ before. Note that there are no changes with respect to the sheep in F.

If in some key κ there is a set $K_i = S' \cup D'$ with $S' \subseteq S$ and $\emptyset \neq D' \subseteq D$ the algorithm proceeds as follows, for every such key (and every such i). For each $j \leq f(\kappa)$, it introduces a new key κ_j in which every $\sigma \in D'$ is replaced by (σ, j). If $S' \neq \emptyset$ then an additional key κ' is added with S' in place of K_i. All these keys inherit $\oplus_i = \bullet$. The key κ itself is removed.

For each of the new keys, the automaton is modified to test that there are no key violations by matches in which every $\bullet$-position is matched with a special dog position.

Let I_1 be the resulting extended constraint instance. If the family, from which the class type (S, D) that was made special was chosen, has further non-special class types, the algorithm restarts from the beginning but with I_1 in place of I. In this manner it constructs a sequence of extended constraint instances $I_1, I_2, \ldots$. However, as the number of unbounded class types decreases in each iteration, the whole process terminates either by reaching an unbounded instance or by making the last class type of some family F special. In the former case the algorithm accepts, in the latter case it removes F from $\mathcal{F}$ by a procedure described next.

Let $F = (S, \emptyset, \mathcal{E})$ be the family that has to be removed. For each dog set $E \in \mathcal{E}$ and each sheep $\sigma \in S$ a new symbol (σ, E) is introduced. These symbols are special symbols and are used to construct special class types from the dog sets in $\mathcal{E}$ which are then put into $\mathcal{C}$. For each $E \in \mathcal{E}$ and each set $K_i \subseteq S \cup E$ of a key κ, the symbols $\sigma \in K_i \cap S$ are replaced by (σ, E). Furthermore, $\oplus_i$ is set to $\circ$.

If any of the resulting keys becomes *nullary*, that is, it does not contain any $\bullet$-positions, its verification is delegated to the automaton $\mathcal{A}$.

If a resulting key κ is *unary* with the only remaining $\bullet$-position j, then the algorithm proceeds as follows. Let $F' = (S', \mathcal{D}', \mathcal{E}')$ be the family of K_j. We distinguish three cases: (1) $K_j \subseteq S'$, (2) $K_j \cap D \neq \emptyset$, for some $D \in \mathcal{D}'$ and (3) $K_j \cap E \neq \emptyset$, for some $E \in \mathcal{E}'$.

In the first case, for each non-empty[12] subset M of K_j and each $D \in \mathcal{D}'$ a new set D_M is introduced with the symbols from D together with new symbols (σ, κ, M), for every $\sigma \in M$. In the automaton these symbols are handled as σ was handled before. However, the automaton checks that a symbol (σ, κ, M) only occurs at places where κ matches and σ n only occurs at places where κ does *not* match. In keys $\kappa' \neq \kappa$ with a position set $K \subseteq S'$, the new symbols are added to K and then it is split with respect to the sets M.

In the second case, for each non-empty subset M of $K_j \cap S$ and each $D \in \mathcal{D}$ a new set D_M is introduced, just as in the first case. Furthermore, for each subset M of K_j with $M \cap D \neq \emptyset$, a set D_M with all symbols from $D - M$ and all symbols (σ, κ, M) with $\sigma \in M$ is added as well. Changes to the automaton and to the keys are analogous to the previous case.

In the third case, sets of the form D_M with $M \subseteq K_j \cap S$ are added as before. Then the algorithm guesses a subset $M \subseteq K_j$ and replaces E by the set of all symbols from $E - M$ and all symbols (σ, κ, M) with $\sigma \in M$. Changes to the automaton and to the keys are again analogous to the previous case.

In all three cases, κ is removed from $\mathcal{K}$ afterwards. This does not do any harm as the new type constraints together with the additional checks by the automaton guarantee that κ is not violated.

This completes the description of the algorithm.

We explain now why this algorithm has the stated properties.

Let $I_0 = I$ denote the input instance for the algorithm. By induction on i it can be shown that for each computation with (at least) i iterations the following holds.

(1) If I is satisfiable then there is a computation for which I_i is satisfiable.

(2) If for some computation I_i is satisfiable then I is satisfiable.

For (1) it suffices to observe that, if I_{i-1} has a solution then it has a solution in some linear set L_0 that the algorithm can choose initially. All modification steps that transform I_{i-1} into I_i are such that a solution of I_{i-1} can be transformed into a solution of I_i. The only modifications might rename some symbols in the solution. For (2) it can be observed that all modifications are such that from a solution to I_i a solution to I_{i-1} can be constructed.

Thus, the algorithm is "locally" sound and complete.

Local soundness immediately yields statement (b).

We already mentioned that the algorithm always terminates either by accepting or by reaching an instance with fewer families. By local completeness we can conclude (c).

Next we show (a), that is, if the algorithm accepts, the current instance I_r at this point has a solution. To see this, we construct a (non-extended) instance from I_r by moving all bounded dog sets E from families F to $\mathcal{C}$. The resulting instance fulfills the prerequisites of Proposition 4.4 and therefore has a solution w. Clearly, w is also a solution for I_r. It should be noted that this solution does not have any sheep in classes of types from sets E. $\square$

Now we are prepared to give a proof of the main result.

Proof of Theorem 4.1. The non-deterministic decision algorithm that tests satisfiability of $\mathrm{FO}^2(\sim, +1)$ formulas φ over data words in the presence of a set $\mathcal{K}$ of key constraints proceeds as follows. First, using the algorithm of the proof of Proposition 4.2, it constructs a constraint instance I. This can be turned into an extended constraint instance by choosing all sets $\mathcal{E}$ empty. Then, by iteratively using the algorithm of Proposition 4.5 it either accepts I (in which case $(\varphi, \mathcal{K})$ is satisfiable) or it constructs a sequence of (extended) constraint instances with a decreasing number of families ending with some instance with empty $\mathcal{F}$ and $\mathcal{K}$. Such an instance only has a non-empty set $\mathcal{C}$ of special

[12]Note that $M = \emptyset$ would yield D itself.

class types and an automaton $\mathcal{A}$ which, besides many other things, tests all former key constraints. Clearly, solutions of such an instance can only have a bounded number of classes and thus, satisfiability can be tested by a non-emptiness test for a finite automaton. $\square$

As the algorithm involves the iterated application of procedures that might result in exponentially larger instances, there does not seem to be an elementary upper bound.

5. LOWER BOUND FOR SATISFIABILITY OF FO$^2(\sim, +1, +3)$ ON DATA WORDS

THEOREM 5.1. *It is* 2-NEXPTIME*-hard to decide whether a FO$^2(\sim, +1, +3)$ formula φ is satisfiable.*

PROOF. The proof is by a reduction from the 2-NEXPTIME-complete problem 2-EXP-corridor tiling. An input to 2-EXP-corridor tiling is a tuple $T = (\Gamma, H, V, \alpha, \omega, 1^n)$.

A *valid $m \times m$-corridor tiling for T* is a mapping $\lambda : [1, m] \times [1, m] \to \Gamma$ such that the following constraints are satisfied:

(1) the bottom row starts with α, i.e., $\lambda(1, 1) = \alpha$;

(2) the top row ends with ω, i.e., $\lambda(m, m) = \omega$;

(3) all vertical constraints are satisfied, i.e., $\forall i \in [1, m)$, $\forall j \in [1, m]$, $(\lambda(i, j), \lambda(i + 1, j)) \in V$; and,

(4) all horizontal constraints are satisfied, i.e., $\forall i \in [1, m]$, $\forall j \in [1, m)$, $(\lambda(i, j), \lambda(i, j + 1)) \in H$.

2-EXP-corridor tiling consists of all T with a valid $2^{2^n} \times 2^{2^n}$-corridor tiling.

The main idea of the reduction is to build a formula Ψ, which checks, that every data string w that satisfies Ψ is divided into blocks, where each block encodes a tile γ, a row number r and a column number c. The formula Ψ further checks whether the corridor tiling derived from w in the canonical way fulfills T. The details of the proof are ommited due to lack of space. $\square$

6. CONCLUSION

The decidability result for FO$^2(\sim, +1)$ over data words in the presence of key constraints is only a first step for the study of the corresponding problem on data trees. It should be noted that, as usual, the result also holds for existential monadic second order logic with two variables in the first-order part.

However, many questions remain open:

- What is the exact complexity of satisfiability of FO$^2(\sim, +1)$ over data words in the presence of key constraints?
- How can key constraints be used in proofs of stronger lower bounds?
- Is the problem still decidable in the presence of a linear order?
- Does the result translate to trees?
- What is the exact complexity of the satisfiability problem for FO$^2(\sim, +1)$ over data words without key constraints? In particular:
  - Can the upper bound of Theorem 3.4 be extended in the presence of the +3-relation (and further $+k$-relations)?
  - Can the lower bound of Theorem 5.1 also be shown for FO$^2(\sim, +1)$?
- What about other kinds of integrity constraints?
- Are the results meaningful in the context of automatic verification?

7. REFERENCES

[1] Marcelo Arenas, Wenfei Fan, and Leonid Libkin. On the complexity of verifying consistency of xml specifications. *SIAM J. Comput.*, 38(3):841–880, 2008.

[2] Marcelo Arenas and Leonid Libkin. A normal form for xml documents. *ACM Trans. Database Syst.*, 29:195–232, 2004.

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

[4] M. Bojanczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3), 2009.

[5] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 7–16, 2006.

[6] Claire David, Leonid Libkin, and Tony Tan. On the satisfiability of two-variable logic over data words. Extended abstract to appear in 17th LPAR 2010, LNCS, Springer-Verlag., 2010.

[7] Stéphane Demri and Ranko Lazic. Ltl with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

[8] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

[9] Diego Figueira. Forward-xpath and extended register automata on data-trees. In *ICDT*, pages 231–241, 2010.

[10] Erich Grädel and Martin Otto. On logics with two variables. *Theor. Comput. Sci.*, 224(1-2):73–113, 1999.

[11] Marcin Jurdzinski and Ranko Lazic. Alternating automata on data trees and xpath satisfiability. *CoRR*, abs/0805.0330, 2008.

[12] Maarten Marx. First order paths in ordered trees. In *ICDT*, pages 114–128, 2005.

[13] Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, pages 41–57, 2006.

[14] Anthony Widjaja To. Parikh images of regular languages: Complexity and applications. *CoRR*, abs/1002.1464, 2010.