Expressive power and data complexity of first-order logic over trees

Evgeny Dantsin* Andrei Voronkov†

Abstract

First-order logic over finite structures (FO) is a query language for relational databases; its expressive power and complexity have been well studied in database theory. We study *first-order logic over trees* (FO^{t}) which is a query language for relational databases with tree-like data. While structures in FO have finite universes, structures in FO^{t} may have infinite universes consisting of *trees*. Such trees are represented in FO^{t} by terms.

Although FO^{t} looks more expressive than FO, we prove that they have the same expressive power (under a natural encoding of structures with trees by finite structures). We also prove that FO^{t} has polynomial data complexity, i.e., any query definable in FO^{t} can be evaluated in time polynomial in the size of inputs.

These results imply similar results for nonrecursive Datalog$^{\neg}$ and its extension for dealing with trees. In particular, these two languages have the same expressive power, which shows that the use of trees in nonrecursive Datalog$^{\neg}$ gives no gain in the expressiveness.

1 Introduction

It is well known that first-order logic over finite structures (FO) cannot express many important relations, for example transitive closure, graph connectivity, acyclicity [Aho and Ullman 1979, Fagin 1975, Gaifman 1982]. The standard way to increase its expressive power is to extend first-order logic by various recursion-like operators, such as fixpoint operators [Aho and Ullman 1979, Chandra and Harel 1982, Abiteboul and Vianu 1991, Immerman 1986, Vardi 1982]. Is it possible to increase the expressive power by using recursive data structures instead of recursion-like operators? For example, one could use trees, probably the most natural recursive data structure. In this paper we show that such an approach does not increase the expressiveness.

Main results. *First-order logic over trees* (FO^{t}) can be informally described as the following extension of FO. First, we allow vocabularies to contain function symbols; they play the role of constructors for tree-like data. Second, we extend the class of structures. Recall that a structure in FO has a finite universe. A structure in FO^{t} may have an infinite universe consisting of *trees*; they can be represented by terms.

Any sentence ϕ in FO^{t} expresses a boolean query that maps every structure S (representing a database instance) to *true* or *false*. The main result of this paper is about how to evaluate ϕ on S. Note that variables in ϕ may range over an infinite universe. We prove that each sentence ϕ in FO^{t} can be translated into a sentence ϕ' in FO such that for every structure S,

$$S \models \phi \quad \text{if and only if} \quad \rho S \models \phi',$$

*Department of Computer Science, University of Manchester. On leave from Steklov Institute of Mathematics, St.Petersburg. Supported in part by grants from EPSRC,INTAS and RFBR. Email: `dantsin@cs.man.ac.uk`

†Department of Computer Science, University of Manchester. Email: `voronkov@cs.man.ac.uk`

where ρ is a "natural" encoding of structures with trees by finite structures. Thus, each query expressible in $\mathrm{FO^t}$ can be expressed in FO (under the encoding ρ). The translation also shows that $\mathrm{FO^t}$ has polynomial data complexity, i.e., ϕ on S can be evaluated in time polynomial in the size of S. This follows from the equivalence above because (1) the encoding ρ is computable in polynomial time, and (2) any FO query is computable in time polynomial in the size of inputs (and even in constant time with a polynomial number of processors, [Immerman 1987]).

Note that although $\mathrm{FO^t}$ and FO are very close in the data complexity, there is a huge difference between them in the program complexity, i.e., the complexity of evaluating ϕ on S when ϕ is variable and S is fixed. Namely, the program complexity of FO is polynomial-space [Vardi 1982], while the program complexity of $\mathrm{FO^t}$ is nonelementary [Compton and Henson 1990], see also [Vorobyov 1996, Vorobyov and Voronkov 1998].

Our results are heavily based on *Safety Theorem* which we prove in Section 4. This theorem reduces the evaluation of a sentence whose variables range over an infinite set of trees to the evaluation of another sentence whose variables range over a finite set of trees (cf. quantifier elimination in term algebras [Maĺcev 1961, Belegradek 1988, Maher 1988, Hodges 1993]). Similar techniques were used in [Hull and Su 1994] for the relational data model and in [Paredaens, Van den Bussche and Van Gucht 1995, Benedikt and Libkin 1997] for constraint databases.

Nonrecursive Datalog$^\neg$ with trees. The results on FO and $\mathrm{FO^t}$ imply similar results for nonrecursive Datalog$^\neg$ and its extension for dealing with trees. This extension is obtained from nonrecursive Datalog$^\neg$ by adding interpreted (built-in) relations for handling trees; its expressive power is the same as the expressive power of $\mathrm{FO^t}$. Since nonrecursive Datalog$^\neg$ is equivalent to FO, it follows from our results on FO and $\mathrm{FO^t}$ that the use of trees in nonrecursive Datalog$^\neg$ gives no gain in the expressiveness.

Like $\mathrm{FO^t}$, nonrecursive Datalog$^\neg$ with trees has polynomial data complexity. Also, using the Safety Theorem, we prove the following interesting property of nonrecursive Datalog$^\neg$ with trees: any query definable in nonrecursive Datalog$^\neg$ with trees is a boolean combination of queries definable in its range-restricted fragment.

Organization of paper. Section 2 contains definitions and notation related to FO and $\mathrm{FO^t}$. The main results, Theorem 1 and Theorem 2, are formulated in Section 3. The proof of Theorem 1 is given after the Safety Theorem in Section 4. In Section 5 we extend nonrecursive Datalog$^\neg$ for dealing with trees and in Section 6 consider nonrecursive Prolog.

2 First-order logic over trees

First-order languages. Let $\mathcal{V}$ be a *vocabulary* that consists of a finite number of *function* and *relation* symbols, both may have arity 0. We assume that $\mathcal{V}$ contains at least one relation symbol. The *first-order language* $\mathcal{L}(\mathcal{V})$ is defined as usual, the equality symbol $=$ is part of the language. A *sentence* is a formula without free variables.

Queries definable in FO. Let $\mathcal{V}$ be a vocabulary without function symbols. For such a vocabulary $\mathcal{V}$, a *finite structure* is a pair $S = \langle U, I\rangle$, where U is a nonempty finite set called a *universe* and I is an *interpretation* that maps the relation symbols of $\mathcal{V}$ to relations of respective arities on U. We write $S \models \phi$ if ϕ is true on S.

Note that our definition does not require an ordering on a universe.

A *boolean query* q is a mapping from the set of all finite structures with vocabulary $\mathcal{V}$ to $\{true, false\}$. Since we do not consider nonboolean queries in this paper, we say "query" instead of "boolean query". We say that q is definable in *first-order logic over finite structures* (abbreviated to FO) if there is a sentence $\phi \in \mathcal{L}(\mathcal{V})$ such that for every finite structure S with vocabulary $\mathcal{V}$, the following equivalence holds: q maps S to *true* if and only if $S \models \phi$.

Queries definable in FO^{t}**.** Now let $\mathcal{V}$ be an arbitrary vocabulary, with or without function symbols. Let Ur be a nonempty finite set whose elements will be called *urelements*. We define the set $\mathcal{T}(Ur)$ of *trees* (over Ur) as follows:

1. Each element of Ur is a tree.
2. If $w_1, \ldots, w_k$ are trees and f is a k-ary function symbol of $\mathcal{V}$ then $f(w_1, \ldots, w_k)$ is a tree.

The set of *subtrees* of a tree t is defined as follows: (i) each urelement u has u as the only subtree; (ii) the subtrees of $f(w_1, \ldots, w_k)$ are $f(w_1, \ldots, w_k)$ itself and all subtrees of $w_1, \ldots, w_k$. A *structure with trees* is defined to be a pair $S = \langle \mathcal{T}(Ur), I \rangle$, where Ur is a finite. Thus, the universe of S consists of trees built up from a finite set of urelements by using the tree constructors corresponding to the function symbols of $\mathcal{V}$. Note that if $\mathcal{V}$ contains at least one function symbol then $\mathcal{T}(Ur)$ is an infinite set, otherwise $\mathcal{T}(Ur)$ is Ur. The *interpretation* I is required to satisfy the following conditions.

1. Each k-ary function symbol f of $\mathcal{V}$ (if any) is interpreted by the operation that maps every tuple $\langle w_1, \ldots, w_k \rangle$ of trees to the tree $f(w_1, \ldots, w_k)$.
2. Each relation symbol of $\mathcal{V}$ is interpreted by a *finite* relation on $\mathcal{T}(Ur)$.

By $\mathcal{S}(\mathcal{V})$ we denote the set of all structures with trees for $\mathcal{V}$. If $\mathcal{V}$ contains no function symbols, $\mathcal{S}(\mathcal{V})$ is the set of all finite structures for $\mathcal{V}$. Queries are defined similar to the case of FO. We say that a query q is definable in *first-order logic over trees* (abbreviated to FO^{t}) if there is a sentence $\phi \in \mathcal{L}(\mathcal{V})$ such that for every structure $S \in \mathcal{S}(\mathcal{V})$, the following equivalence holds: q maps S to *true* if and only if $S \models \phi$.

3 Main results

Translation from FO^{t} **into** FO**.** According to our definition, each query definable in FO can be defined in FO^{t}, therefore the expressive power of FO^{t} is not less than the expressive power of FO. We show that, in a sense, the converse holds.

Structures with trees can be represented by finite structures in the following natural way. For each vocabulary $\mathcal{V}$ with or without function symbols, we will define an *encoding* ρ that maps every structure $S \in \mathcal{S}(\mathcal{V})$ to a finite structure $\rho S \in \mathcal{S}(\rho\mathcal{V})$, where $\rho\mathcal{V}$ is some vocabulary without function symbols.

First, we define the vocabulary $\rho\mathcal{V}$. Let $\mathcal{V}$ consist of function symbols $f_1, \ldots, f_l$ and relation symbols $R_1, \ldots, R_m$. The vocabulary $\rho\mathcal{V}$ consists of the same relation symbols $R_1, \ldots, R_m$ and new relation symbols $F_1, \ldots, F_l$ that correspond to the function symbols. Each symbol F_i has the arity equal to the arity of f_i plus 1. Let R_i^S denote the set of all trees that occur in the interpretation of R_i in S.

Now we define the structure ρS. Its universe is the union of two sets, namely (1) the set of all urelements and (2) the set of all subtrees of the trees of $R_1^S \cup \ldots \cup R_k^S$. Note that the universe of ρS

is *finite*. The interpretation of $c_1, \dots, c_k$ and $R_1, \dots, R_m$ remains the same as in S. Each symbol F_i is interpreted in ρS as follows: $\langle u, v_1, \dots, v_n \rangle \in F_i$ if and only if u is $f_i(v_1, \dots, v_n)$.

We emphasize that ρ is not simply an encoding that changes a functional vocabulary to a relational one. It is more important that ρ also changes an *infinite* universe to a *finite* one. Speaking informally, the universe of ρS is restricted to the set that consists of the urelements of S and all subtrees of the trees appearing in the relations of S.

The encoding ρ induces a natural mapping from the queries over structures with trees to the queries over finite structures. Namely, a query q over structures with trees is mapped to a query q' such that for every S,the following equivalence holds: $q(S) = \mathit{true}$ if and only if $q'(\rho S) = \mathit{true}$. We show that if a query q is definable in FO^{t} then the query q' is definable in FO.

Theorem 1 *There is an algorithm that translates each sentence $\phi \in \mathcal{L}(\mathcal{V})$ into a sentence $\phi' \in \mathcal{L}(\rho\mathcal{V})$ such that for every structure $S \in \mathcal{S}(\mathcal{V})$, we have $S \models \phi$ if and only if $\rho S \models \phi'$.*

Proof. We prove this theorem in Section 4 after Theorem 4 (*Safety Theorem*) in which we construct the required sentence ϕ'. $\square$

Complexity of evaluation of FO^{t} queries. To evaluate a query definable in FO^{t}, we evaluate a sentence $\phi \in \mathcal{L}(\mathcal{V})$ on a structure $S \in \mathcal{S}(\mathcal{V})$. In the context of databases, the size of ϕ is usually much less than the size of S. Therefore we are mostly interested in the *data complexity*, i.e., the complexity of evaluating ϕ on S when ϕ is fixed and S is variable. We prove that the data complexity of FO^{t} is polynomial.

Note that although S may have an infinite universe, we can represent S by a finite object. Namely, S is uniquely determined by (1) its vocabulary, (2) its set of urelements, and (3) its relations (which are finite by definition). Our complexity results will hold for any "reasonable" representations of this object. For example, urelements can be represented by natural numbers, trees can be represented by ground terms built up from these numbers by using function symbols, etc.

Theorem 2 *For any vocabulary $\mathcal{V}$, any sentence $\phi \in \mathcal{L}(\mathcal{V})$, and any structure $S \in \mathcal{S}(\mathcal{V})$, the value of ϕ on S can be computed in time polynomial in the size of S.*

Proof. Theorem 1 reduces the evaluation of ϕ on S to the following computational tasks: (1) compute ρS; (2) evaluate ϕ' on ρS. The first task can be solved in polynomial time. This follows from the fact that under any "reasonable" representation of trees, the encoding ρ preserves the size up to a polynomial (for any tree, the total size of all its different subtrees is polynomial in its size). The second problem, i.e., evaluating a first-order sentence on a finite structure when the sentence is fixed and the structure is variable, is well-known: this problem is in AC^0 [Immerman 1987]. Thus, both tasks can be solved in polynomial time. $\square$

4 Quantifier elimination for first-order logic over trees

In this section we prove *Safety Theorem* which is in fact a theorem on quantifier elimination for FO^{t}. Then we prove Theorem 1 which was formulated in the previous section. To keep the proof short we make a simplifying assumption on the vocabulary $\mathcal{V}$: it contains only one binary function symbol f

and one unary relation symbol R. The generalization to arbitrary vocabularies is straightforward. The class of all structures with trees for the simplified vocabulary $\mathcal{V}$ is denoted by K.

Also for simplification, we say that two first-order formulas are *equivalent* if they are equivalent on all structures of K. Likewise, we call a first-order formula *valid* if it is valid on every structure of K.

Our main technical tool is the notion of *safe formulas* introduced below, cf. [Ullman 1988,1989]. The purpose of this section is to prove that every first-order sentence is equivalent to a boolean combination of safe sentences. This statement (*Safety Theorem*) plays the same role as quantifier elimination in term algebras, see e.g. [Hodges 1993].

Notation. For a structure $S \in K$, we denote by R^S the interpretation of R in S; thus R^S is a finite set of trees. We extend the vocabulary $\mathcal{V}$ by adding two unary function symbols l (from "left") and r (from "right"), and a unary relation symbol Ur. Their interpretation in the structures of K is defined as follows: (1) $l(u) = r(u) = u$ for any urelement u; (2) $l(f(t_1, t_2)) = t_1$ and $r(f(t_1, t_2)) = t_2$ for all t_1, t_2. The relation symbol Ur is interpreted as the set of all urelements. The extended vocabulary is denoted by $\mathcal{V}^+$.

We introduce some notation for terms and formulas of the extended vocabulary. We will leave out parentheses in successive applications of l and r, for example, we write lrt instead of $l(r(t))$. Any term of $\mathcal{V}^+$ not using f, i.e., built up from variables by using zero or more applications of l and r, will be called an *$(l$-$r)$-term.* Lowercase Greek letters α, β, γ stand for sequences, maybe empty, of l's and r's. Such sequences denote compositions of l and r in $(l$-$r)$-terms. For example, if $\alpha = lrr$ then αx denotes the term $lrrx$. We write $G(t_1, \ldots, t_n)$ for any term obtained by zero or more applications of f to $(l$-$r)$-terms $t_1, \ldots, t_n$. For example, $G(x, ly)$ may stand for the term $f(f(x, ly))$ or the term $f(f(ly, x), x)$. We use this notation to emphasize that $t_1, \ldots, t_n$ are $(l$-$r)$-terms. Note that a term $G(t_1, \ldots, t_n)$ has no subterms of the form $lf(s_0, s_1)$ or $rf(s_0, s_1)$.

4.1 Safe formulas and quasi-literals

Definition 1 ($(l$-$r)$-depth) For every free variable x in a formula A of the vocabulary $\mathcal{V}^+$, we define the *$(l$-$r)$-depth* of x in A to be the maximal length of α such that αx appears in A.

For example, the $(l$-$r)$-depth of x in the formula $y \neq lx \wedge R(f(llrx, f(rry, x)))$ is 3.

Definition 2 (safe formula) The class of *safe formulas* of the vocabulary $\mathcal{V}^+$ is defined by induction as follows:

1. For any term $G(t_1, \ldots, t_n)$ of $\mathcal{V}^+$, the formulas $R(G(t_1, \ldots, t_n))$ and $Ur(G(t_1, \ldots, t_n))$ are safe.

2. If both A and B are safe, then $A \wedge B$ is safe.

3. If both A and B are safe and every free variable in B is a free variable in A, then $A \wedge \neg B$ is safe.

4. If A is safe, $s_1 \neq t_1, \ldots, s_n \neq t_n$, $n \geq 0$ are disequalities, and x is a variable such that (i) $s_1, \ldots, s_n, t_1, \ldots, t_n$ are $(l$-$r)$-terms whose variables are free in A, (ii) x occurs in every disequality $s_i \neq t_i$ for $i = 1, \ldots, n$, (iii) the $(l$-$r)$-depth of x in A is 0, then the formula

$$\exists x(s_1 \neq t_1 \wedge \ldots s_n \neq t_n \wedge A)$$

is safe.

The intuition behind safe formulas is expressed by the following lemma.

Lemma 3 (finite number of valuations) *Let $A(\bar{x}, \bar{y})$ be a safe formula in which every variable of $\bar{x}$ has (l-r)-depth 0. Let $S \in K$ be a structure. Then for every valuation τ of $\bar{y}$, there exists only a finite number of valuations ν of $\bar{x}$ such that $S, \tau, \nu \models A(\bar{x}, \bar{y})$.*

Proof. By straightforward induction on the definition of safe formulas, using the fact that both R^S and the set of urelements are finite in every structure $S \in K$. □

It is convenient to think of safe formulas as a generalization of atoms. It is also convenient to introduce an analog of a literal.

Definition 3 (quasi-literal) A *quasi-literal* is either of the following formulas: (1) a safe formula or its negation; (2) a formula $t_1 = t_2$ or $t_1 \neq t_2$, where t_1 and t_2 are (l-r)-terms.

4.2 Safety theorem

Theorem 4 (safety theorem) *There is an algorithm that transforms any formula A of the vocabulary $\mathcal{V}^+$ into an equivalent boolean combination A' of quasi-literals. If A is a sentence then A' is a boolean combination of safe sentences.*

The proof will be given below, after a series of lemmas which show how to eliminate "unsafety" in subformulas of A.

Lemma 5 (decreasing (l-r)-depth) *Let $A(x)$ be a quasi-literal with a free occurrence of x. Then for every variables x_0 and x_1 not occurring in $A(x)$, the formula $A(f(x_0, x_1))$ is equivalent to a boolean combination B of quasi-literals. In addition, B has the following property: if x has a nonzero (l-r)-depth d in $A(x)$ then both (l-r)-depths of x_0 and x_1 in B are strictly less than d.*

Proof. Without loss of generality we assume that no trivial equalities $x = \alpha x$ or disequalities $x \neq \alpha x$ occur in $A(x)$. The formula $A(f(x_0, x_1))$ may not be a quasi-literal for two reasons. First, the terms $l(f(x_0, x_1))$ and $r(f(x_0, x_1))$ may appear. In this case we replace them by x_0 and x_1 respectively. Second, f may appear in equalities $f(x_0, x_1) = t$ and disequalities $f(x_0, x_1) \neq t$. Then by our assumption x does not occur in t. We replace such equalities and disequalities using the following equivalences:

$$
\begin{array}{rcl}
f(x_0, x_1) = t & \equiv & \neg Ur(t) \ \wedge \ x_0 = lt \ \wedge \ x_1 = rt, \\
f(x_0, x_1) \neq t & \equiv & Ur(t) \ \vee \ (\neg Ur(t) \ \wedge \ (x_0 \neq lt \ \vee \ x_1 \neq rt)).
\end{array}
$$

Using the distributivity laws and transformations $\exists y(B_1 \vee B_2) \rightarrow \exists y B_1 \vee \exists y B_2$ we can turn the resulting formula into a boolean combination of quasi-literals. The assertion about the depth of x_0, x_1 is straightforward. □

Lemma 6 (negated formulas) *Let A be a formula*

$$\alpha_1 x \neq \beta_1 y_1 \wedge \ldots \wedge \alpha_n x \neq \beta_n y_n \wedge \neg B_1(x) \wedge \ldots \wedge \neg B_m(x),$$

where (i) the (l-r)-depth of x in every $B_i(x)$ is 0, (ii) for every i the term $\alpha_i x$ is different from $\beta_i y_i$, and (iii) each B_j is a safe formula containing a free occurrence of x. Then $\exists x A$ is valid.

Proof. Let $\bar{y}$ denote the list of all free variables of $\exists x A$. If we show that for every valuation τ of $\bar{y}$ there exists a valuation ν for x such that $S, \tau, \nu \models A$, then we are done. To show this, take any such valuation τ. Define the *height* of a tree by induction as follows: the height of any urelement is 0, and the height of $f(x, y)$ is the maximum of the heights of x and y plus 1. By Lemma 3, there exists a finite number of valuations ν' for x such that $S, \tau, \nu' \models B_1(x) \vee \ldots \vee B_m(x)$. Let h_1 be the maximum of the heights of $\nu'(x)$ for all such valuations ν', and h_2 be the maximum height of all trees in $\tau(\bar{y})$, or 0 if $\bar{y}$ is empty. Define $H = h_1 + h_2 + 1$. Further, define h_3 be the maximum length of all α_i and β_i.

It is not hard to argue that there exists a tree t such that (i) for every sequence γ of the symbols l and r of the length at most h_3 the height of $\gamma(t)$ is greater than or equal to H; moreover (ii) for every two different sequences γ_1, γ_2 of the symbols l and r of the length at most h_3, it holds that $\gamma_1(t)$ is different from $\gamma_2(t)$.

Define ν to be the valuation that maps x into such t. We prove that $S, \tau, \nu \models A$.

1. We prove $S, \tau, \nu \models \neg B_i(x)$. Indeed, by our construction $S, \tau, \nu' \models B_i(x)$ only for valuations ν' such that the height of $\nu'(x)$ is at most h_1, but the height of $\nu(x)$ is at least H.

2. We prove $S, \tau, \nu \models \alpha_i x \neq \beta_i y_i$. Consider two cases:

   (a) x is different from y_i. Then the height of $\tau(\beta_i y_i)$ is at most h_2 but the height of $\nu(\alpha_i x)$ is at least H.

   (b) x coincides with y_i. In this case we have to show $S, \tau, \nu \models \alpha_i x \neq \beta_i x$. Since by the condition of the lemma α_i is different from β_i, then by our construction $\alpha_i(\nu(x))$ is different from $\beta_i(\nu(x))$.

□

Lemma 7 (elimination of "unsafe" quantifiers) *Any formula $\exists x(A_1(x) \wedge \ldots \wedge A_k(x))$, where $A_1(x), \ldots, A_k(x)$ are quasi-literals, is equivalent to a boolean combination of quasi-literals.*

Proof. If the (l-r)-depth of x in $A_1(x) \wedge \ldots \wedge A_k(x)$ is positive, we can replace this formula by the equivalent one

$$\exists x(Ur(x) \wedge A_1(x) \wedge \ldots \wedge A_k(x)) \vee \exists x_0 \exists x_1(A_1(f(x_0, x_1)) \wedge \ldots \wedge A_k(f(x_0, x_1)))$$

and then using Lemma 5 make the (l-r)-depths of x_0, x_1 in this formula less than the (l-r)-depth of x in the original formula. So we can assume that the (l-r)-depth of x in the original formula is zero. Then our conjunction consists of only four types of quasi-literals.

1. Equalities $x = t$. Since the (l-r)-depth of x is 0, we can assume that x does not occur in t.

2. Disequalities $x \neq t$. As above, we can assume that x does not occur in t.

3. Safe formulas $B(x)$, which contain a free occurrence of x and maybe other free variables.

4. Negated safe formulas $\neg B(x)$, which contain a free occurrence of x and maybe other free variables.

If the conjunction contains $x = t$, we eliminate $\exists x$ by using the equivalence $\exists x(x = t \wedge C(x)) \equiv C(t)$. Therefore, we assume that the conjunction contains no equalities. If the conjunction consists of only disequalities and negated safe formulas, the formula $\exists x(A_1 \wedge \ldots \wedge A_k)$ is valid by Lemma 6. It remains to consider the case where at least one safe formula $B(x)$ occurs in the conjunction.

Now our conjunction $A_1 \wedge \ldots \wedge A_k$ contains members A_i of only three types: $B(x)$, $\neg B(x)$ and $x \neq t$. We call members of the first type *safe*, and the other members *unsafe*. A free variable u in the conjunction is called *unsafe* if it occurs in only unsafe members. Clearly, if there are no unsafe variables, then the conjunction is safe and we are done. We show how to get rid of unsafe variables.

Let u be an unsafe variable. The formula $\exists x(A_1 \wedge \ldots \wedge A_k)$ has the form

$$\begin{array}{l} \exists x(C(x) \wedge \neg D_1(x,u) \wedge \ldots \wedge \neg D_k(x,u) \wedge \\ \quad x \neq \alpha_1 u \wedge \ldots \wedge x \neq \alpha_m u \wedge E(x)), \end{array} \tag{1}$$

where $C(x)$ is a safe formula and u has no occurrences in $C(x)$ or $E(x)$. Denoting by $D(x,u)$ the formula

$$D_1(x,u) \vee \ldots \vee D_k(x,u) \vee x = \alpha_1 u \vee \ldots \vee x = \alpha_m u,$$

we rewrite (1) as

$$\exists x(C(x) \wedge \neg D(x,u) \wedge E(x)). \tag{2}$$

Using the tautology

$$\forall x(C(x) \supset \neg D(x,u)) \vee \exists x(C(x) \wedge D(x,u)), \tag{3}$$

we turn formula (2) into the equivalent formula

$$\begin{array}{l} \exists x(C(x) \wedge \forall x(C(x) \supset \neg D(x,u)) \\ \quad \wedge \neg D(x,u) \wedge E(x)) \end{array} \tag{4}$$

$$\vee$$

$$\begin{array}{l} \exists x(C(x) \wedge \exists x(C(x) \wedge D(x,u)) \\ \quad \wedge \neg D(x,u) \wedge E(x)) \end{array} \tag{5}$$

First, consider subformula (4). Using Modus Ponens, we turn it into the equivalent formula $\forall x(C(x) \supset \neg D(x,u)) \wedge \exists x(C(x) \wedge E(x))$. Since $\forall x(C(x) \supset \neg D(x,u))$ is equivalent to $\neg \exists x(C(x) \wedge D(x,u))$, this formula is a boolean combination of $\exists x(C(x) \wedge E(x))$, in which u does not occur, and $\exists x(C(x) \wedge D(x,u))$. Using the definition of $D(x,u)$, we can rewrite the last formula into the equivalent formula

$$\begin{array}{l} \exists x(C(x) \wedge D_1(x,u)) \vee \ldots \vee \\ \exists x(C(x) \wedge D_k(x,u)) \vee \\ \exists x(C(x) \wedge x = \alpha_1 u) \vee \ldots \vee \\ \exists x(C(x) \wedge x = \alpha_m u). \end{array}$$

This formula is equivalent to

$$\begin{array}{l} \exists x(C(x) \wedge D_1(x,u)) \vee \ldots \vee \\ \exists x(C(x) \wedge D_k(x,u)) \vee C(\alpha_1 u) \vee \ldots \vee C(\alpha_m u), \end{array}$$

The last formula is a boolean combination of formulas which either have less quantifiers than $\exists x(A_1 \wedge \ldots \wedge A_k)$ or in which u is safe.

Second, consider subformula (5). Using the definition of $D(x,u)$, we rewrite (5) as

$$\begin{array}{l} \exists x(C(x) \wedge \exists x(C(x) \wedge D_1(x,u)) \\ \qquad \wedge \neg D(x,u) \wedge E(x)) \;\; \vee \\ \dots \\ \vee \;\; \exists x(C(x) \wedge \exists x(C(x) \wedge D_k(x,u)) \\ \qquad \wedge \neg D(x,u) \wedge E(x)) \\ \vee \;\; \exists x(C(x) \wedge \exists x(C(x) \wedge x = \alpha_1 u) \\ \qquad \wedge \neg D(x,u) \wedge E(x)) \;\; \vee \\ \dots \\ \vee \;\; \exists x(C(x) \wedge \exists x(C(x) \wedge x = \alpha_m u) \\ \qquad \wedge \neg D(x,u) \wedge E(x)). \end{array}$$

This formula is equivalent to

$$\begin{array}{l} \exists x(C(x) \wedge \exists x(C(x) \wedge D_1(x,u)) \\ \qquad \wedge \neg D(x,u) \wedge E(x)) \;\; \vee \\ \dots \\ \vee \;\; \exists x(C(x) \wedge \exists x(C(x) \wedge D_k(x,u)) \\ \qquad \wedge \neg D(x,u) \wedge E(x)) \\ \vee \;\; \exists x(C(x) \wedge C(\alpha_1 u) \\ \qquad \wedge \neg D(x,u) \wedge E(x)) \;\; \vee \\ \dots \\ \vee \;\; \exists x(C(x) \wedge C(\alpha_m u) \\ \qquad \wedge \neg D(x,u) \wedge E(x)) \end{array}$$

in which all occurrences of u are safe. This completes the proof. □

Proof of Theorem 4. The input formula A is transformed as follows. First, we eliminate all occurrences of $\forall x$ in A by replacing them by $\neg\exists x\neg$. Next, we eliminate all occurrences of f in equalities and disequalities as in the proof of Lemma 5. Likewise, we get rid of all subterms of the form $lf(t_0,t_1)$ and $rf(t_0,t_1)$. Now, each atomic subformula of A is a quasi-literal.

We continue the proof by induction on A. The cases when A is a conjunction, disjunction, or negation are obvious. It remains to consider the case when A is $\exists xB$, where B is a boolean combination of quasi-literals. By moving negations inwards and using the distributivity laws for conjunction and disjunction, we can turn B into a formula

$$\bigvee_i \bigwedge_j B_{i,j},$$

where each $B_{i,j}$ is a quasi-literal. Then we can replace $\exists xB$ by the equivalent disjunction

$$\bigvee_i \exists x \bigwedge_j B_{i,j}.$$

Now we apply Lemma 7 to turn every subformula $\exists x \bigwedge_j B_{i,j}$ into a boolean combination of quasi-literals.

The second part of the theorem is much easier. Suppose that A is a sentence. Apply the above algorithm to turn A into a boolean combination of quasi-literals $A_1,\dots,A_k$. Since the algorithm introduces no new free variables, every A_i is closed, therefore is a safe sentence or its negation (because an equality $s=t$ or a disequality $s \neq t$ cannot be closed).

4.3 Proof of Theorem 1

Using the Safety Theorem, we now prove Theorem 1 (more precisely, we prove its reformulation for the simplified vocabulary $\mathcal{V}$).

For every structure $S \in K$, by D^S we denote the union of two sets: the set of all urelements of S and the set of all subtrees of the trees belonging to R^S. According to our definition of structures with trees, D^S is finite.

Lemma 8 (bounded quantifiers) *Let S be a structure in K. Let $A(\bar{x})$ be a safe formula of the vocabulary $\mathcal{V}$ in which $\bar{x}$ are all free variables in A. For any valuation ν of $\bar{x}$, if $S, \nu \models A(\bar{x})$ then ν maps $\bar{x}$ to D^S.*

Proof. By straightforward induction on the definition of safe formulas. □

It is easy to see that this lemma shows that safe formulas can be evaluated on ρS instead of S. To this end, we extend σ by a new monadic relation symbol D with the following interpretation: for every structure S, the symbol D is interpreted by the set D^S of trees. *Bounded quantifiers* $(\forall x \in D)A$ and $(\exists x \in D)A$ denote $\forall x(D(x) \supset A)$ and $\exists x(D(x) \wedge A)$ respectively. For every formula A, we denote by $A|_D$ the formula obtained from A by replacement of all quantifiers $\forall x$ and $\exists x$ by the respective bounded quantifiers $\forall x \in D$ and $\exists x \in D$.

Lemma 9 *If A is a safe formula of the vocabulary $\mathcal{V}$, then A is equivalent to $A|_D$.*

Proof. By induction on the definition of safe formulas. The only nontrivial case is when A is $\exists x B$, where B is safe. By Lemma 8, $\exists x B$ is equivalent to $(\exists x \in D)B$. By the induction hypothesis, the latter formula is equivalent to $(\exists x \in D)B|_D$, i.e. to $A|_D$. □

This lemma immediately implies Theorems 1.

5 Nonrecursive Datalog$^\neg$ with trees

In Section 2 we extended FO for dealing with tree-like data. In this section we define a similar extension for another traditional query language. Namely, we extend *nonrecursive Datalog*$^\neg$ by adding interpreted (built-in) relations for handling trees. We show that this extension has the same expressive power as FO^{t}. Since nonrecursive Datalog$^\neg$ is equivalent to FO, Theorem 1 implies that the use of trees in nonrecursive Datalog$^\neg$ does not increase the expressiveness. This partly explains why lists and trees are not so widely used in nonrecursive query languages as other collection types.

Like FO^{t}, nonrecursive Datalog$^\neg$ with trees has polynomial data complexity. Another interesting property is that any query definable in nonrecursive Datalog$^\neg$ with trees is a boolean combination of queries definable in its range-restricted fragment.

5.1 Extension of nonrecursive Datalog$^\neg$

Nonrecursive Datalog$^\neg$ and its variations are defined in e.g. [Abiteboul, Hull and Vianu 1995]. We consider the version in which clauses are not required to be range-restricted. Programs in nonrecursive Datalog$^\neg$ are viewed as queries that map *database instances* to $\{true, false\}$ (for simplicity, we consider only boolean programs).

There is a natural correspondence between database instances and finite structures. For each database instance D, let S_D be a finite structure such that (1) the universe of S_D consists of all elements occurring in the relations of D, and (2) the relations of S_D coincide with the relations of D. Conversely, for each structure S, there is a database instance D_S such that (1) the domain of D_S is the universe of S, and (2) the relations of D_S are the relations of S plus the unary relation consisting of all elements of the domain. Under this correspondence, the following equivalence holds.

Proposition 10 (e.g. [Abiteboul et al. 1995]) *A query q is definable in nonrecursive Datalog$^\neg$ if and only if q is definable in* FO.

To define *nonrecursive Datalog$^\neg$ with trees*, we generalize the notion of database instances and the correspondence between database instances and structures.

Let S be a structure with trees for a vocabulary $\mathcal{V}$. Assume that $\mathcal{V}$ contains function symbols $f_1, \ldots, f_l$. Let $\{u_1, \ldots, u_n\}$ be the set of urelements of S. We define a database instance D_S as follows. The domain of D_S is the infinite set of all trees of S. The set of relations of D_S consists of (1) all relations of S and (2) new relations $Ur, F_1, \ldots, F_l$. The unary relation Ur is the set $\{u_1, \ldots, u_n\}$ of urelements. The relations $F_1, \ldots, F_l$ correspond to the function symbols $f_1, \ldots, f_l$; each F_i has the arity equal to the arity of f_i plus 1 (cf. the encoding ρ in Section 3). Unlike other relations in D_S, the relations $F_1, \ldots, F_l$ are *infinite*. Namely, for each relation F_i and for each tree u in the domain, we have $\langle u, v_1, \ldots, v_n\rangle \in F_i$ if and only if u is $f_i(v_1, \ldots, v_n)$.

Now, given a vocabulary $\mathcal{V}$ with function symbols $f_1, \ldots, f_l$, we obtain *nonrecursive Datalog$^\neg$ with trees* from nonrecursive Datalog$^\neg$ by adding *interpreted* relations $Ur, F_1, \ldots, F_l$. Each input for nonrecursive Datalog$^\neg$ with trees is a *database instance over trees*, i.e., a database instance D_s corresponding to some structure S for $\mathcal{V}$.

The proof of the equivalence between nonrecursive Datalog$^\neg$ and FO ([Abiteboul et al. 1995]) can be adapted for database instance over trees and structures with trees, thus the following equivalence holds.

Proposition 11 *A query q is definable in nonrecursive Datalog$^\neg$ with trees if and only if q is definable in* FO^{t}.

5.2 Expressive power and data complexity of nonrecursive Datalog$^\neg$ with trees

In Section 3 we defined the encoding ρ which encodes structure with trees by finite structures. It is easy to define a similar encoding (denoted by the same letter ρ) which encodes database instances over trees by ordinary database instances. Under this encoding, nonrecursive Datalog$^\neg$ with trees is equivalent to ordinary nonrecursive Datalog$^\neg$.

Theorem 12 *For each program π in nonrecursive Datalog$^\neg$ with trees, there is a program π' in nonrecursive Datalog$^\neg$ such that for every database instance D over trees, the answer of π on D is the same as the answer of π' on ρD.*

Proof. Follows from Proposition 10, Proposition 11 and Theorem 1. □

Although a database instance over trees may have an infinite domain, we can represent it by a finite object (similar to the representation of structures with trees, see Section 3). The following theorem shows that nonrecursive Datalog$^\neg$ with trees has polynomial data complexity.

Theorem 13 *For any program π in nonrecursive Datalog$^\neg$ with trees, the answer of π on D, where D is a database structure over trees, can be computed in time polynomial in the size of D.*

Proof. Like the encoding ρ for structures, the encoding ρ for database instances preserves the size up to a polynomial. Therefore, the proof of this theorem is similar to the proof of Theorem 2. $\square$

5.3 Range-restricted fragment

First, we need to adapt the standard notion of range-restricted nonrecursive Datalog$^\neg$ [Abiteboul et al. 1995] for the extension for trees. Let C be a clause Datalog$^\neg$ with trees. The notion of a *safe variable* in C is defined as follows.

1. Every variable occurring in a positive literal L with the predicate symbol different from the relation symbols $F_1, \ldots, F_l$ in the body of C is safe.
2. If $F_i(u, v_1, \ldots, v_n)$ is a positive literal in C, then
   (a) If u is safe, then so are $v_1, \ldots, v_n$.
   (b) If all $v_1, \ldots, v_n$ are safe, then so is u.

A clause C is called *range-restricted* if every variable in C is safe. A program is called *range-restricted* if all its clauses are range-restricted.

Lemma 14 *For each safe sentence ϕ in FO^{t}, there exist a program π in range-restricted nonrecursive Datalog$^\neg$ such that for every structure S, the value of ϕ on S is equal to the answer of π on D_S.*

Proof. By induction on the definition of safe formulas. $\square$

Theorem 15 *Each query definable in nonrecursive Datalog$^\neg$ with trees is a boolean combination of queries definable in its range-restricted fragment.*

Proof. Let q be a query definable in nonrecursive Datalog$^\neg$ with trees. By Proposition 11, q is definable by a sentence ϕ in FO^{t}. By Theorem 4, ϕ is a boolean combination of safe sentences $\phi_1, \ldots, \phi_n$. By Lemma 14, the queries definable by $\phi_1, \ldots, \phi_n$ can be defined in range-restricted nonrecursive Datalog$^\neg$ with trees. $\square$

6 Nonrecursive Prolog

Nonrecursive Prolog and its range-restricted fragment is defined in e.g. [Lloyd 1987, Vorobyov and Voronkov 1998]. There exist polynomial-time transformations between nonrecursive Prolog and first-order logic over trees, see e.g. [Vorobyov and Voronkov 1998] for details. Therefore, our main results can be reformulated for Prolog as follows:

Theorem 16 *For any program π in nonrecursive Prolog, the answer of π on S, where S is a structure with trees, can be computed in time polynomial in the size of S.*

Theorem 17 *Each query definable in nonrecursive Prolog is a boolean combination of queries definable in range-restricted nonrecursive Prolog.*

References

Abiteboul, S., Hull, R. and Vianu, V. [1995], *Foundation of Databases*, Addison-Wesley Publishing Co.

Abiteboul, S. and Vianu, V. [1991], 'Datalog extensions for database queries and updates', *Journal of Computer and System Sciences* **43**, 62–124.

Aho, A. and Ullman, J. [1979], Universality of data retrieval languages, *in* 'ACM Symposium on Principles of Programming Languages', pp. 110–117.

Belegradek, O. [1988], Model theory of locally free algebras (in Russian), *in* 'Model Theory and its Applications', Vol. 8 of *Trudy Instituta Matematiki*, Nauka, Novosibirsk, pp. 3–24. Translated in [Belegradek 1999].

Belegradek, O. [1999], 'Model theory of locally free algebras', *American Mathematical Society Translations* **195**.

Benedikt, M. and Libkin, L. [1997], Languages for relational databases over interpreted structures, *in* 'PODS 1997. Proceedings of the Sixteenth ACM-SIGMOD-SIGART Symposium on Principles of Database Systems', Tucson, Arizona, pp. 87–98.

Chandra, A. and Harel, D. [1982], 'Structure and complexity of relational queries', *Journal of Computer and System Sciences* **25**, 99–128.

Compton, K. and Henson, C. [1990], 'A uniform method for proving lower bounds on the computational complexity of logical theories', *Annals of Pure and Applied Logic* **48**, 1–79.

Fagin, R. [1975], 'Monadic generalized spectra', *Zeitschrift für Mathematical Logic und Grundlagen der Mathematik* **21**, 89–96.

Gaifman, H. [1982], On local and nonlocal properties, *in* J. Stern, ed., 'Logic Colloquium'81', North Holland, pp. 105–135.

Hodges, W. [1993], *Model theory*, Cambridge University Press.

Hull, R. and Su, J. [1994], 'Domain independence and the relational calculus', *Acta Informatica* **31**, 513–524.

Immerman, N. [1986], 'Relational queries computable in polynomial time', *Information and Control* **68**, 86–104.

Immerman, N. [1987], 'Languages that capture complexity classes', *SIAM Journal of Computing* **16**, 760–778.

Lloyd, J. [1987], *Foundations of Logic Programming*, 2nd edn, Springer Verlag.

Maher, M. [1988], Complete axiomatizations of the algebras of finite, rational and infinite trees, *in* 'Proc. IEEE Conference on Logic in Computer Science (LICS)', pp. 348–357.

Maĺcev, A. [1961], 'On the elementary theories of locally free universal algebras', *Soviet Mathematical Doklady* **2**(3), 768–771.

Paredaens, J., Van den Bussche, J. and Van Gucht, D. [1995], First-order queries on finite structures over the reals, *in* 'LICS'95', pp. 79–87.

Ullman, J. [1988,1989], *Database and Knowledge-Base Systems*, Vol. I,II, Computer Science Press.

Vardi, M. [1982], The complexity of relational query languages, *in* 'Proc. 14th ACM STOC', San Francisco, pp. 137–146.

Vorobyov, S. [1996], An improved lower bound for the elementary theories of trees, *in* M. McRobbie and J. Slaney, eds, 'Automated Deduction — CADE-13', Vol. 1104 of *Lecture Notes in Computer Science*, New Brunswick, NJ, USA, pp. 275–287.

Vorobyov, S. and Voronkov, A. [1998], Complexity of nonrecursive logic programs with complex values, *in* 'PODS'98', ACM Press, Seattle, Washington, pp. 244–253.