MAX-PLANCK-INSTITUT FÜR INFORMATIK

Complexity of Nonrecursive Logic Programs with Complex Values

Sergei Vorobyov and Andrei Voronkov

MPI–I–97–2–010 November 1997

I N F O R M A T I K

Im Stadtwald
D 66123 Saarbrücken
Germany

Author's Address

Sergei Vorobyov: Max-Planck Institut für Informatik, Im Stadtwald, D-66123, Saarbrücken, Germany, sv@mpi-sb.mpg.de, http://www.mpi-sb.mpg.de/~sv.

Andrei Voronkov: Computing Science Department, Uppsala University, Box 311, S-75105, Uppsala, Sweden, voronkov@csd.uu.se, http://www.csd.uu.se/~voronkov.

Publication Notes

Part of this report was written while the second author visited MPI in 1997. The second author is partially supported by a TFR grant.

Acknowledgements

We thank Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Gabriel Kuper.

Abstract

We investigate complexity of the SUCCESS problem for logic query languages with complex values: check whether a query defines a nonempty set. The SUCCESS problem for recursive query languages with complex values is undecidable, so we study the complexity of nonrecursive queries. By complex values we understand values such as trees, finite sets, and multisets. Due to the well-known correspondence between relational query languages and datalog, our results can be considered as results about relational query languages with complex values. The paper gives a complete complexity classification of the SUCCESS problem for nonrecursive logic programs over trees depending on the underlying signature, presence of negation, and range restrictedness. We also prove several results about finite sets and multisets.

Keywords

Logic programming, complexity of query languages, complex values.

Contents

1 Introduction

A number of complexity results have been established for logic query languages. They are surveyed in (Schlipf 1995, Dantsin, Eiter, Gottlob & Voronkov 1997). The major themes in these results are the complexity and expressive power of extensions of datalog: the logic query language for describing relation over tuples of simple, non-structured objects. Due to the well-known correspondence between datalog and relational algebra, the complexity results about nonrecursive datalog can be restated as results about relational query languages, for example, SQL-92 or its fragments.

New relational query languages, for example SQL-3, extend traditional languages in several directions. One of them is the introduction of complex values, like sets. There is no uniform convention on how complex values should be handled in relational query languages. Indeed, the introduction of any new type of values requires addition of new operations on the corresponding algebras.

In logic query languages, stemming from logic programming, there seems to be a uniform viewpoint on how complex values should be treated. There are two major approaches.

1. In *constraint logic programming* (Maher 1992, Maher 1993) and *constraint databases* (Kanellakis, Kuper & Revesz 1995) any value is identified by the set of constraints true on this value. The addition of a new type of values requires the addition of new constraint predicates. A similar approach to relational query languages was also considered in (Benedikt & Libkin 1997).

2. Another approach to adding complex values, which can be called *structural*, requires that values be represented by means of their structure. For example, to represent sets one may enrich the language with constant $\emptyset$ to denote the empty set and the set constructor $\{s|t\}$ denoting the addition of an element s to the set t. Then the set $\{t_1, \ldots, t_n\}$ will be denoted by the term $\{t_1| \ldots \{t_n|\emptyset\} \ldots\}$. The only changes to the semantics of logic programming are the changes in the treatment of equality, since new predicate symbols are not free constructors. Such an approach is considered in a number of papers, for example (Gallier & Raatz 1989, Kuper 1990, Beeri, Naqvi, Schmueli & Tsur 1991, Schmueli, Tsur & Zaniolo 1992, Dovier, Omodeo, Pontelli & Rossi 1996, Dantsin & Voronkov 1997*b*, Dantsin & Voronkov 1997*a*).

Of course, a combination of the two approaches is also possible.

This paper studies complexity of nonrecursive query answering in logic databases with complex values. Nonrecursive queries in logic databases are represented by nonrecursive logic programs. Among nonrecursive logic programs we distinguish range-restricted ones, as of special interest for databases. By complex values we understand various versions of trees, finite multisets and sets. For example, we give a complete classification of complexity for nonrecursive logic programs with trees depending on the signature, range-restrictedness and presence of negation.

It is possible to reformulate our complexity results in terms of some relational query languages with complex values, using suitable algebraic formalizations corresponding to nonrecursive logic programs, see, e.g., (Abiteboul & Beeri 1995).

Results of this paper show that nonrecursive query languages for complex values are highly intractable. It will be interesting to investigate these classes in terms of fixed-parameter complexity similar to the analysis done in (Papadimitriou & Yannakakis 1997, Vardi 1995).

We briefly mention some results on the complexity of recursive logic programs. These (and other results) are surveyed in (Dantsin et al. 1997).

Definite programs. For definite programs without function symbols, the following results are known. The SUCCESS problem is *DEXPTIME*-complete for recursive programs (Vardi 1982, Immerman 1986) and *PSPACE*-complete for nonrecursive programs (Vardi 1982, Dantsin 1986, Immerman 1987). With function symbols, it is r.e.-complete (Andréka & Németi 1978, Tärnlund 1977).

Normal programs. For logic programs with negation, several nonmonotonic semantics exist. In the case of *stratified* programs most of these semantics coincide with the *perfect model semantics*. For this semantics the following results are known. The SUCCESS problem for programs without function symbols is *PSPACE*-complete (Vardi 1982, Dantsin 1986, Immerman 1987) in the case of nonrecursive programs and *NEXPTIME*-complete in the case of recursive programs (Apt & Blair 1988*b*). In the case of arbitrary function symbols, (Apt & Blair 1988*a*) prove that the SUCCESS problems for programs with n levels of stratified negation is Σ_n^0-complete. (Blair, Marek & Schlipf 1995) address the expressive power of locally stratified recursive programs and show that every hyperarithmetic (that is Δ^1_1) set is definable by a stratified program over a perfect model.

This paper studies the complexity of nonrecursive logic programs over complex values, such as trees, finite sets and multisets. We do not study the expressive power of languages with complex values. We do not consider recursive query languages or aggregation.

2 Preliminaries

Logic and signatures. By first-order logic we mean first-order logic with equality. Formulas are constructed using all standard connectives, except the equivalence $\equiv$. The equality predicate is denoted $\approx$. We denote by $\Phi\left[\psi_i/\varphi_i\right]_{i=1}^{m}$ the formula obtained from Φ by simultaneous replacement of all occurrences of subformulas φ_i by formulas ψ_i.

Our results will depend on the signature in which programs are written. We shall only consider *functional signatures*, consisting of finite or countable sets of function symbols.

If a signature Σ consists of k constants, l unary function symbols, and m function symbols of arity ≥ 2, we shall denote such a signature by (k, l, m). We shall also use ordinals and wildcards in the notation for signatures. For example, $(\omega, \_, 3)$ denotes any signature with infinitely many constants, any number of unary function symbols, and 3 function symbols of arity ≥ 2. Similarly, $(< \omega, 0, 0)$ denotes a signature with any finite number of constants and no function symbols of arity ≥ 1.

> **Proviso.** *We always assume that signatures have at least two symbols, including at least one constant.*

Other signatures can also be considered but are not interesting for our aims.

Logic programming. We assume knowledge of standard facts about semantics of logic programming. They may be found in, e.g., (Apt 1990) or (Lloyd 1987). *Clauses* will be written as

$$P(t_1, \ldots, t_n) \leftarrow L_1, \ldots, L_m.$$

We shall consider both *normal clauses* where each L_i is a literal and *definite clauses* where each L_i is an atomic formula. We assume that the equality predicate $\approx$ does not occur in clauses. A *normal (*respectively, *definite) program* is a finite set of normal (respectively, definite) clauses. A clause is called *range-restricted*, if all variables occurring in the clause also occur in a positive literal in the body.

Instead of considering a logic program $\mathcal{P}$, it will be convenient to work with *Clark's completion* (Clark 1978) of $\mathcal{P}$, that is a set of *predicate definitions* of the form

$$
\begin{array}{l}
P_0(\bar{x}_0) \equiv \Phi_0, \\
\cdots \\
P_n(\bar{x}_n) \equiv \Phi_n,
\end{array}
\tag{1}
$$

where all P_i are different predicates and each Φ_i is a first-order formula whose free variables are contained in $\bar{x}_i$.

For example, the set of two clauses

$$
\begin{array}{l}
P(a, x) \leftarrow R(x, z) \\
P(f(y), x) \leftarrow \neg R(x, z)
\end{array}
$$

can be rewritten as the following definition:

$$
P(u, x) \equiv \exists z(u \approx a \wedge R(x, z)) \vee \exists y \exists z(u \approx f(y) \wedge \neg R(x, z))
$$

We only consider *nonrecursive* logic programs. In Clark's completion (1) of any nonrecursive logic program, each formula Φ_i may only contain predicate symbols among $\approx, P_0, \ldots, P_{i-1}$. When a program is definite, the formulas Φ_i are constructed using only $\exists, \vee, \wedge$.

Semantics and term algebras. There are several approaches to defining semantics of normal logic programs. However, in the nonrecursive case the semantics can be described by using the so-called term algebras.

Definition 2.1 The *term algebra* of a signature Σ, denoted $TA(\Sigma)$, is the algebra in which the carrier set is the set of ground terms of Σ, any ground term is interpreted by itself and any two distinct ground terms are nonequal. □

Such algebras are also called *absolutely free algebras* (Maĺcev 1961*a*). In view of our proviso on signatures, term algebras considered in this paper have at least two elements.

> **Proviso.** *In this article we consider two formulas equivalent, if they are equivalent in any term algebra with at least two elements.*

By $Th(TA(\Sigma))$ we denote the *first-order theory* of $TA(\Sigma)$, i.e., the set of all sentences of the signature Σ with equality true in $TA(\Sigma)$. The decidability of $Th(TA(\Sigma))$ for finite signatures was proved in (Maĺcev 1961*b*, Maĺcev 1961*a*) by quantifier elimination. Later (Kunen 1987b, Maher 1988, Hodges 1993) proved it again for the case of finite and infinite signatures, also by using quantifier elimination. The $PSPACE$-completeness of $Th(TA(\geq 2,0,0))$ is due to (Stockmeyer & Meyer 1973, Stockmeyer 1977). The $LATIME(2^{O(n)})$-completeness[1] of $Th(TA(1,2,0))$ is due to (Volger 1983b). (Kunen 1987a) asserted $PSPACE$-completeness of $Th(TA(\omega,\omega,\omega))$, but it was proved non-elementary recursive with a linearly growing stack of twos as a lower bound in (Vorobyov 1996).

Nonrecursive definitions (1) can be regarded as explicit definitions of new predicates over a term algebra $TA(\Sigma)$, where Σ contains all symbols occurring in (1). The predicate P_0 is defined directly in terms of equality $\approx$, and each P_{i+1} is defined in terms of $P_0,\dots,P_i$. Thus, by eliminating explicit definitions, each P_i is explicitly defined by a formula of $Th(TA(\Sigma))$. This gives a straightforward semantics to nonrecursive logic programs. However, this semantics may depend on Σ (this fact is usually called *domain-dependence*). For example, the formula $\forall x(x \approx a \vee x \approx b)$ is true in $TA(\{a,b\})$ and false in $TA(\{a,b,c\})$. It is well known (Abiteboul, Hull & Vianu 1995) that the semantics of P_i is domain-independent in the case of range-restricted clauses.

For any nonrecursive program $\mathcal{P}$, we shall refer to the extension of $TA(\Sigma)$ by new predicates P_i explicitly defined by $\mathcal{P}$, as to the *perfect model* of $\mathcal{P}$ in Σ (Przymusinski 1988, Van Gelder 1988, Lifschitz 1988).

The SUCCESS problem. By the SUCCESS(Σ) *problem* for a class of logic programs $\mathcal{C}$ of the signature Σ we mean the following decision problem: given a logic program $\mathcal{P} \in \mathcal{C}$ and a goal G, does G succeed with respect to $\mathcal{P}$, where the definitions are understood as over $TA(\Sigma)$? When G succeeds with respect to $\mathcal{P}$ we can also say that the pair $(\mathcal{P}, G)$ defines a nonempty query.

Instead of checking whether G succeeds with respect to $\mathcal{P}$, we can introduce a new predicate success, add the clause success $\leftarrow G$ to $\mathcal{P}$, and ask whether success is true in the prefect model of the new program. This means that the last predicate P_n in (1) is success and we have to check whether the sentence defined by success with respect to (1) is true in $TA(\Sigma)$. The complexity of the SUCCESS problem is similar to what is called the *combined complexity* in (Vardi 1982).

[1] $LATIME(2^{O(n)})$ is the class of problems solvable by alternating Turing machines in linear exponential time with linear number of alternations.

A standard form of logic programs. Since we assume at least two symbols in the signature, we may assume to have terms or sequences of terms representing natural numbers less than or equal to some fixed number N in the size $O(\log N)$. Let 0 and 1 be any two distinct ground terms. We encode any natural number $n < N$ as a sequence of terms 0 and 1 that forms the binary representation of n. For example, if $N = 6$, we need only 3 digits to represent numbers up to 6 and the number 2 can be represented as the sequence of arguments $(0, 1, 0)$ to some predicate P.

Now, we show that we can restrict ourselves to definitions (1) in which P_{i+1} is defined solely in terms of P_i, i.e., Φ_{i+1} uses only P_i and $\approx$ as predicate symbols. Without loss of generality we can assume that all P_i have the same number of arguments since we can always add dummy arguments to P_i. Consider predicates $Q_0, \ldots, Q_n$ having the same number of arguments as P_i plus enough arguments to represent natural numbers up to n. Our intention is to make $Q_i(j, \bar{x})$ equivalent to $P_j(\bar{x})$ for all $j \leq i$.

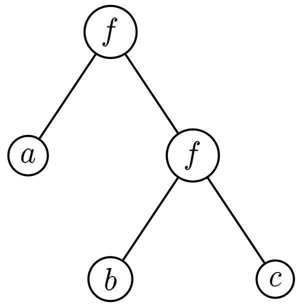
Introduce formulas $\Psi_i = \Phi_i \Big[Q_j(j, \bar{t}) / P_j(\bar{t}) \Big]_{j=1}^{n}$ and define Q_i as follows:

$$
\begin{array}{l}
Q_0(u, \bar{x}) \equiv u \approx 0 \wedge \Psi_0; \\
Q_{i+1}(u, \bar{x}) \equiv \\
\qquad (u \approx i+1 \wedge \Psi_{i+1}) \vee \\
\qquad (\bigvee_{j=1}^{i} u \approx j \wedge Q_i(u, \bar{x})) \;\; \text{for all } 0 < i < n; \\
\mathtt{success} \equiv \exists \bar{x} Q_n(n, \bar{x}).
\end{array}
$$

The equivalence of $Q_i(j, \bar{x})$ and $P_j(\bar{x})$ is straightforward. Therefore (see the definition of $\mathtt{success}$ in terms of Q_n), we conclude that the $\mathsf{SUCCESS}$ problem for the definitions of predicates P_i is equivalent to that for the definitions of predicates Q_i.

Note that the definitions of Q_i's can be constructed from the definitions of P_i's in polynomial time.

Complex values. Here we briefly consider what kind of data are represented by first-order terms. The discussion of multisets and sets is postponed until Section 8. Terms of standard (unsorted) first-order logic represent *trees*. For example, the term $f(a, f(b, c))$ represents the following tree



Trees may have an arbitrary depth.

Some formalizations of complex values in databases use *embedded tuples*. Embedded tuples correspond to sorted first-order logic, where every function symbol belongs to some sort. If all atomic domains are finite, every sort contains only a *finite* number of values, while trees allow one to construct an infinite number of values from a finite atomic domain. We shall consider both typed and untyped versions.

We pay special attention to signatures consisting of unary function symbols. Terms in such signatures can be understood as representing *lists* of atomic values. For example, the term $f(g(g(h(a))))$ represents the list $[f, g, g, h]$.

3 Nonrecursive logic programs and term algebras

In this section we show that the SUCCESS problem for nonrecursive logic programs with negation is polynomial time equivalent to the underlying theory of term algebra. Thus, results on the complexity of $Th(TA(\Sigma))$ for various Σ are directly applicable to nonrecursive logic programming.

Theorem 3.1 For every signature Σ with at least two symbols, the theory $Th(TA(\Sigma))$ is polynomial-time equivalent to $\mathsf{SUCCESS}(\Sigma)$. □

A polynomial time reduction of $Th(TA(\Sigma))$ to $\mathsf{SUCCESS}(\Sigma)$ is well-known and described in, e.g., (Lloyd 1987). It can be traced to Clark's completion. Thus, it remains to prove polynomial time reducibility of $\mathsf{SUCCESS}(\Sigma)$ to $Th(TA(\Sigma))$.

The proof is divided into two lemmas. An auxiliary Lemma 3.2 describes a succinct way to write polynomially short formulas (which otherwise would be exponentially long) by replacing multiple positive occurrences of the same predicate with just one such occurrence. Lemma 3.3 gives a required reduction.

Lemma 3.2 Given a quantifier-free formula Φ containing m positive occurrences

$$P(\bar{t}_i), \text{ for } i = 1, \ldots, m$$

of the same predicate P with different parameters $\bar{t}_i$, and no negative occurrences P, one can construct in polynomial time an equivalent formula Δ containing just one positive occurrence and no negative occurrences of P.

Proof. Take fresh variables $x_1, y_1, \ldots, x_m, y_m$ and consider the formula

$$\Phi' = \Phi\Big[x_i \approx y_i / P(\bar{t}_i)\Big]_{i=1}^{m}.$$

Let us show that Φ is equivalent to

$$\Psi = \exists x_1 y_1 \ldots x_m y_m \Big(\Phi' \wedge \bigwedge_{i=1}^{m} (x_i \approx y_i \supset P(\bar{t}_i))\Big).$$

We must prove that for every interpretation ν of the free variables of Φ (or, equivalently, of Ψ), $\nu(\Phi)$ is true iff $\nu(\Psi)$ is true.

Let $\nu(\Phi)$ be true. We may choose equal values for x_i, y_i if $P(\bar{t}_i)$ is true, and different values for x_i, y_i otherwise. Then $\nu(\Psi)$ is true.

Suppose $\nu(\Psi)$ is true for some interpretation ν of its free variables. Then for this interpretation and some values of $x_1, y_1, \ldots, x_m, y_m$ the following subformulas of Ψ are true:

$$\Phi' = \Phi\Big[x_i \approx y_i / P(\bar{t}_i)\Big]_{i=1}^{m}, \tag{2}$$

$$\bigwedge_{i=1}^{m}(x_i \approx y_i \Rightarrow P(\bar{t}_i)). \tag{3}$$

Recall that Φ is positive in $P(\bar{t}_i)$, hence, by construction, Φ' is positive in $x_i \approx y_i$. Therefore, Φ' is monotone in $x_i \approx y_i$. This and (3) imply that $\Phi'\Big[P(\bar{t}_i)/x_i \approx y_i\Big]_{i=1}^{m}$ is true. But this formula coincides with Φ.

The formula Ψ still contains m occurrences of P. Take fresh variables u, v, and $\bar{z}$ (vector of length equal to the arity of P). The subformula

$$\psi = \bigwedge_{i=1}^{m}(x_i \approx y_i \supset P(\bar{t}_i))$$

of Ψ containing m occurrences of P is equivalent to

$$\delta = \forall uv\bar{z}\Big(\bigvee_{i=1}^{m}(u \approx x_i \wedge v \approx y_i \wedge \bar{z} \approx \bar{t}_i) \supset (u \approx v \supset P(\bar{z}))\Big),$$

which contains just one occurrence of P. The proof of the equivalence of ψ and δ is a routine.

Finally, let Δ be obtained from Ψ by replacement of the occurrence of ψ with δ. Clearly, Δ is equivalent to Φ, can be constructed in polynomial time, and contains just one positive and no negative occurrences of P, as needed. $\square$

We are ready to prove

Lemma 3.3 $\mathsf{SUCCESS}(\Sigma)$ for nonrecursive normal programs is polynomial time reducible to $Th(TA(\Sigma))$.

Proof. As described in Section 2, instead of a nonrecursive logic program we can consider a set of explicit definitions

$$
\begin{aligned}
&P_0(\bar{x}) \equiv \Delta_0,\\
&\dots\\
&P_k(\bar{x}) \equiv \Delta_k,
\end{aligned}
$$

such that each Δ_i contains only the predicate P_{i-1} and equality.

Denote by $\mathfrak{M}$ the perfect model of $\mathcal{P}$. We must demonstrate that one can construct in time polynomial in the size of $\mathcal{P}$ a sentence Φ such that

$$TA(\Sigma) \vdash \Phi \;\Leftrightarrow\; \mathfrak{M} \models \exists\bar{x} P_k(\bar{x}). \tag{4}$$

Since $\mathcal{P}$ is nonrecursive, the predicate P_k is *explicitly defined* in terms of $P_{k-1}, \dots, P_1$, and equality; thus, ultimately, can be explicitly defined in terms of equality only. Therefore, we can write down an explicit definition for P_k and existentially quantify it. The resulting sentence Φ of $TA(\Sigma)$ will satisfy (4). The only drawback of this reduction is that it is *exponential*. This is because an explicit definition of P_i may contain several occurrences of the predicate P_{i-1} with different arguments, which gives an exponential blow-up.

Let us first modify the program $\mathcal{P}$ into a new program $\mathcal{P}'$, in which every predicate P_i will get an additional argument meaning "the value of P_i" or "the value of $\neg P_i$". With this transformation we will get rid of negative occurrences of P_i in the bodies of definitions.

Transform the definition for P_0 in $\mathcal{P}$ into:

$$
\begin{aligned}
&P'_0(0,\bar{x}) \equiv \Delta_0,\\
&P'_0(1,\bar{x}) \equiv \neg\Delta_0,
\end{aligned}
$$

and then combine them into just one definition:

$$P'_0(v,\bar{x}) \equiv (v \approx 0 \lor v \approx 1) \land (v \approx 0 \supset \Delta_0) \land (v \approx 1 \supset \neg\Delta_0)$$

(recall that Δ_0 does not contain occurrences of P_i by definition).

To transform the definition $P_{i+1}(\bar{x}) \equiv \Delta_{i+1}$ of $\mathcal{P}$, where Δ_{i+1} may contain both positive and negative occurrences of P_i, proceed as follows. Obtain the body Δ'_{i+1} by replacing every positive occurrence of $P_i(\bar{t})$ with $P'_i(0,\bar{t})$, and every negative occurrence of $P_i(\bar{s})$ with $\neg P'_i(1,\bar{s})$. Note that Δ'_{i+1} contains *only positive occurrences* of P'_i.

Obtain Δ''_{i+1} by negating Δ'_{i+1} and replacing every occurrence of $P'_i(v,\bar{t})$ with $\neg P'_i((v+1)\ mod\ 2,\bar{t})$. Note that Δ''_{i+1} obtained that way also contains *only positive occurrences* of P'_i.

Now write the resulting definition for P'_{i+1}:

$$P'_{i+1}(v,\bar{x}) \equiv (v \approx 0 \lor v \approx 1) \land (v \approx 0 \supset \Delta'_{i+1}) \land (v \approx 1 \supset \Delta''_{i+1}).$$

Denote the program obtained that way by $\mathcal{P}'$ and its perfect model by $\mathfrak{M}'$. Clearly, the program $\mathcal{P}'$ may be constructed in polynomial time and satisfies the following property easily provable by induction:

$$\mathfrak{M} \models \exists \bar{x} P_k(\bar{x}) \ \Leftrightarrow \mathfrak{M}' \models \exists \bar{x} P'_k(0,\bar{x}).$$

By Lemma 3.2, we may rewrite in polynomial time the program $\mathcal{P}'$ into a new program $\mathcal{P}''$ in such a way that the body of each definition for P'_{i+1} contains *just one positive occurrence* of P'_i. Now the unfolding of the explicit definition of P'_k in terms of equality by using the program $\mathcal{P}''$, will yield in polynomial time a formula Φ satisfying (4), as required. $\Box$

4 Programs without function symbols

The following fact has been observed in, e.g., (Stockmeyer & Meyer 1973, Stockmeyer 1977, Kunen 1987*a*):

Theorem 4.1 The theory $Th(TA(\_, 0, 0))$ is $PSPACE$-complete. $\square$

This fact and Theorem 3.1 give

Theorem 4.2 $\mathsf{SUCCESS}(\_, 0, 0)$ is $PSPACE$-complete for nonrecursive logic programs. $\square$

In order to characterize the complexity of special cases of programs without function symbols, we prove one result that is probably a folklore in logic programming/deductive database community.

Lemma 4.3 $\mathsf{SUCCESS}(\_, 0, 0)$ is $PSPACE$-hard for nonrecursive range-restricted definite logic programs.

Proof. We shall use reduction from the theory $Th(TA(2, 0, 0))$. Two constants of this theory will be denoted by 0 and 1. Given any formula $\varphi(\bar{x})$ of $TA(2, 0, 0)$, we shall construct a logic program $\mathcal{P}$ defining the predicate $P_\varphi(\bar{x}, y)$ such that $TA(2, 0, 0) \models \varphi(\bar{t})$ if and only if $P_\varphi(\bar{t}, 1)$ is true in the perfect model of $\mathcal{P}$; and $TA(2, 0, 0) \models \neg\varphi(\bar{t})$ if and only if $P_\varphi(\bar{t}, 0)$ is true in the perfect model of $\mathcal{P}$. We define P_φ by induction on φ, leaving details for the reader. First, we define predicates $T_\neg$, $T_\wedge$ and $P_{x\approx y}$, defining truth tables for $\neg$, $\wedge$ and truth of equalities $x \approx y$, respectively:

$$\begin{array}{lll} T_\neg(0,1) \leftarrow, & T_\wedge(0,0,0) \leftarrow, & P_{x\approx y}(0,0,1) \leftarrow, \\ T_\neg(1,0) \leftarrow, & T_\wedge(0,1,0) \leftarrow, & P_{x\approx y}(0,1,0) \leftarrow, \\ & T_\wedge(1,0,0) \leftarrow, & P_{x\approx y}(1,0,0) \leftarrow, \\ & T_\wedge(1,1,1) \leftarrow, & P_{x\approx y}(1,1,1) \leftarrow . \end{array}$$

Without loss of generality we can assume that $\varphi(\bar{x})$ is constructed from equalities $s \approx t$ using $\neg, \wedge$ and $\exists$. We define truth for non-atomic formulas as follows:

$$\begin{array}{l} P_{\neg\varphi}(\bar{x}, y) \leftarrow \\ \qquad P_\varphi(\bar{x}, z), \\ \qquad T_\neg(z, y); \end{array}$$

$$
\begin{aligned}
&P_{\varphi_1 \wedge \varphi_2}(\bar{x}, z) \leftarrow \\
&\qquad P_{\varphi_1}(\bar{x}_1, y_1), \\
&\qquad P_{\varphi_2}(\bar{x}_2, y_2), \\
&\qquad T_{\wedge}(y_1, y_2, z); \\
&P_{\exists y \varphi}(\bar{x}, 0) \leftarrow \\
&\qquad P_{\varphi}(\bar{x}, 0, 0), \\
&\qquad P_{\varphi}(\bar{x}, 1, 0); \\
&P_{\exists y \varphi}(\bar{x}, 1) \leftarrow \\
&\qquad P_{\varphi}(\bar{x}, y, 1).
\end{aligned}
$$

Evidently, this program is nonrecursive, definite, range-restricted and can be constructed in time polynomial in the size of φ. □

By Theorem 4.2 and Lemma 4.3 we have

Theorem 4.4 $\mathsf{SUCCESS}(\_, 0, 0)$ is *PSPACE*-complete for the following classes of nonrecursive programs: (i) definite programs; (ii) normal programs; (iii) definite range-restricted programs; (iv) normal range-restricted programs. □

5 Programs with binary function symbols

The complexity of the SUCCESS problem for definite programs (range-restricted or not) is characterized by the following theorem proved in (Dantsin & Voronkov 1997*b*, Dantsin & Voronkov 1997*c*).

Theorem 5.1 SUCCESS$(_,_,\geq 1)$ is *NEXPTIME*-complete for nonrecursive definite programs. The same result holds for range-restricted nonrecursive definite programs. □

To characterize the complexity of arbitrary nonrecursive logic programs, define functions $e_0(n) = n$, $e_{k+1}(n) = 2^{e_k(n)}$, and $e_\infty(n) = e_n(0)$. Recall that a problem is called *elementary recursive*, if it can be decided within time bounded by $e_k(n)$ for some fixed $k \in \omega$. Denote by *NONELEMENTARY*$(f(n))$ the class of problems with lower and upper time bounds of the form $e_\infty(f(cn))$ and $e_\infty(f(dn))$ for some $c, d > 0$.

The following result is proved in (Vorobyov 1996):

Theorem 5.2 The theories $Th(TA(\_,\_,\geq 1))$ are not elementary recursive with the lower bound $e_\infty(cn)$ for some $c > 0$. □

The standard quantifier elimination for $TA(\_,\_,\geq 1)$ (Maĺcev 1961*b*, Maĺcev 1961*a*, Hodges 1993) gives the upper bound $e_\infty(dn)$ of the same kind. Hence, the theory of $TA(\_,\_,\geq 1)$ is in *NONELEMENTARY*(n).

Theorems 3.1 and 5.2 give

Theorem 5.3 SUCCESS$(_,_,\geq 1)$ is in *NONELEMENTARY*(n) for nonrecursive normal programs. □

6 Programs with unary function symbols

In this section we prove two complexity results for definite and normal programs in monadic signatures.

6.1 Definite programs

Surprisingly, the SUCCESS problem has the same complexity for nonrecursive definite programs also in the absence of binary function symbols.

It is easy to prove that for definite programs the SUCCESS(Σ) problem does not depend on Σ, see, e.g., (Falashi, Levi, Martelli & Palamidessi 1989). Hence, for definite programs we can always assume that Σ consists of the symbols occurring in the program (plus one constant if the program contains no constants).

Theorem 6.1 SUCCESS($\_, \geq 2, 0$) is *NEXPTIME*-complete for nonrecursive definite programs.

The proof will be given below, after a series of lemmas. To prove *NEXPTIME*-hardness, we shall use the reduction of the TILING problem known to be *NEXPTIME*-complete, see, e.g., (Papadimitriou 1994, page 501).

TILING is the problem of covering the square of size $2^n \times 2^n$ by tiles (squares of size 1×1). There is a finite set $\{f_1, \dots, f_K\}$ of *tiles* and there are two binary relations `on` and `to` defined on the tiles. Tiles f_i and f_j are said to be *vertically compatible* if $\texttt{on}(f_i, f_j)$ holds and, similarly, *horizontally compatible* if $\texttt{to}(f_i, f_j)$ holds. A *tiling of the rectangle of size $m \times n$* is a function f from $\{1, \dots, m\} \times \{1, \dots, n\}$ into $\{f_1, \dots, f_K\}$ such that:

1. $f(i, j)$ and $f(i + 1, j)$ are vertically compatible, for all $1 \leq i < m$ and $1 \leq j \leq n$;

2. $f(i, j)$ and $f(i, j + 1)$ are horizontally compatible, for all $1 \leq i \leq m$ and $1 \leq j < n$.

We also say that such f is a tiling *with f_i at the top left corner* if $f(1, 1)$ is f_i. The TILING problem is defined as follows.

> Given a set $\{f_0, \dots, f_K\}$ of tiles, compatibility relations `on` and `to`, and a number N (written in unary notation), whether there is a tiling of the square of size $2^N \times 2^N$ with f_0 at the top left corner.

The reduction we describe is a polynomial-time algorithm that transforms every instance I of the TILING problem into a nonrecursive definite program $\mathcal{P}$ such that I has a tiling if and only if `success` succeeds with respect to $\mathcal{P}$. We think of tiles $f_1, \ldots, f_K$ in I as all unary function symbols of Σ. Let c be any fixed constant in Σ and N be any fixed positive integer.

We shall use the reverse Polish notation for unary terms in the signature Σ. For example, the term $f_1(f_2(f_3(c)))$ will be written as $cf_3f_2f_1$. Thus, any term with the constant c can be written as cW, where W is a word on the alphabet Σ. We shall identify any word V on Σ with the term cV. The constant c then corresponds to the empty word, denoted ε.

We shall encode rectangles $m \times n$ consisting of tiles in the following way. The rectangle

$$m\text{ tiles}\left\{\overbrace{\begin{array}{|c|ccc|c|}\hline g_1^1 & \cdots & \cdots & \cdots & g_1^n \\ \hline \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \hline g_m^1 & \cdots & \cdots & \cdots & g_m^n \\ \hline \end{array}}^{n\text{ tiles}}\right.$$

will be represented by the word

$$g_1^1 \cdots g_1^n g_2^1 \cdots\cdots\cdots g_{m-1}^n g_m^1 \cdots g_m^n.$$

The compatibility relations `on` and `to` are represented in $\mathcal{P}$ by the corresponding predicates. Namely, $\mathcal{P}$ contains the clauses

$\mathtt{on}(cf_i, cf_j) \leftarrow$
$\mathtt{to}(cf_l, cf_m) \leftarrow$

for all pairs of compatible tiles.

Before representing the TILING problem, we show how to represent concatenation of words of the length 2^i for all $i \leq 2N$ using a definite program of size polynomial in N.

Consider the definite program $\mathcal{P}_1$ consisting of the clauses

$$\mathtt{conc}_1(x, xf_i, y, yf_j, z, zf_if_j) \leftarrow$$

for all $1 \leq i, j < K$ and the clause

$$\begin{aligned} &\mathtt{conc}_{n+1}(x_4, z_4, z_7, v_7, x_3, v_8) \leftarrow \\ &\qquad \mathtt{conc}_n(x_1, y_1, y_2, z_2, x_3, z_3), \\ &\qquad \mathtt{conc}_n(x_1, y_1, y_2, z_2, x_4, z_4), \\ &\qquad \mathtt{conc}_n(z_5, u_5, u_6, v_6, z_7, v_7), \\ &\qquad \mathtt{conc}_n(z_5, u_5, u_6, v_6, z_3, v_8). \end{aligned}$$

for all $1 < n \leq 2N$.

The *difference* between words U and V, denoted $U - V$, is the word defined as follows. If V is a prefix of U, i.e., $VW = U$ for some word W, then $U - V = W$, otherwise the difference is undefined.

Lemma 6.2 Let $r_1, r_2, s_1, s_2, t_1, t_2$ be ground terms of the signature Σ. Then the atom

$$\mathtt{conc}_n(r_1, r_2, s_1, s_2, t_1, t_2)$$

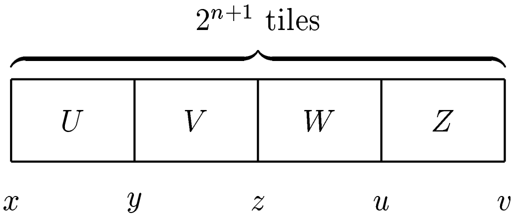
is true in the perfect model of $\mathcal{P}_1$ if and only if

1. r_1 is a prefix of r_2, s_1 is a prefix of s_2, t_1 is a prefix of t_2; and
2. $r_2 - r_1$ and $s_2 - s_1$ are of length 2^{n-1} and $t_2 - t_1$ is of length 2^n; and
3. $(r_2 - r_1)(s_2 - s_1) = t_2 - t_1$.

Proof. For $n = 1$ the proof is straightforward because $\mathtt{conc}_1$ is defined by the clauses

$$\mathtt{conc}_1(x, xf_i, y, yf_j, z, zf_if_j) \leftarrow .$$

Suppose now $n > 1$. By induction hypothesis, we assume that the statement holds for $\mathtt{conc}_n$ and prove that it holds for $\mathtt{conc}_{n+1}$. The proof is illustrated by the following picture:



$\Rightarrow$ We have to prove that $\texttt{conc}_{n+1}(r_1, r_2, s_1, s_2, t_1, t_2)$ implies conditions (1–3) of the lemma. Suppose $\texttt{conc}_{n+1}(r_1, r_2, s_1, s_2, t_1, t_2)$. Then there exist terms $x_1, y_1, y_2, z_2, z_3, z_5, u_5, u_6, v_6$ such that the following conjunction is true:

$$\begin{array}{l}
\texttt{conc}_n(x_1, y_1, y_2, z_2, t_1, z_3) \wedge \\
\texttt{conc}_n(x_1, y_1, y_2, z_2, r_1, r_2) \wedge \\
\texttt{conc}_n(z_5, u_5, u_6, v_6, s_1, s_2) \wedge \\
\texttt{conc}_n(z_5, u_5, u_6, v_6, z_8, t_2)
\end{array}$$

We shall use properties of $\texttt{conc}_n$ given by the induction hypothesis.

Since $\texttt{conc}_n(x_1, y_1, y_2, z_2, r_1, r_2) \wedge \texttt{conc}_n(z_5, u_5, u_6, v_6, s_1, s_2)$, there exist words U, V and W, Z of the length 2^{n-1} such that $r_2 = r_1UV$, $y_1 = x_1U$, $z_2 = y_2V$ and $u_5 = z_5W$, $v_6 = u_6Z$ and $s_2 = s_1WZ$. Then we have $\texttt{conc}_n(x_1, x_1U, y_2, y_2V, t_1, z_3)$. This implies $z_3 = t_1UV$. Then we have $\texttt{conc}_n(z_5, z_5W, u_6, u_6Z, t_1UV, t_2)$. This implies $t_2 = t_1UVWZ$. Now it is easy to check that conditions (1–3) hold.

$\Leftarrow$ Suppose that words $r_1, r_2, s_1, s_2, t_1, t_2$ satisfy conditions (1–3) of the lemma. Let U, V be words of length 2^{n-1} such that $r_2 = r_1UV$ and W, Z be words of length 2^{n-1} such that $s_2 = s_1WZ$. Note that we have $t_2 = t_1UVWZ$. Consider the following ground instance of the clause defining $\texttt{conc}_{n+1}$:

$$\begin{array}{l}
\texttt{conc}_{n+1}(r_1, r_1UV, s_1, s_1WZ, t_1, t_1UVWZ) \leftarrow \\
\quad \texttt{conc}_n(c, cU, c, cV, t_1, t_1UV), \\
\quad \texttt{conc}_n(c, cU, c, cV, t_1, r_1, r_1UV), \\
\quad \texttt{conc}_n(c, cW, c, cZ, s_1, s_1WZ), \\
\quad \texttt{conc}_n, (c, cW, c, cZ, t_1UV, t_1UVWZ).
\end{array}$$

Since arguments of $\texttt{conc}_n$ satisfy conditions (1–3) of the lemma, by the induction hypothesis the body of the clause is true. Therefore the head is also true. But the head is $\texttt{conc}_{n+1}(r_1, r_2, s_1, s_2, t_1, t_2)$.

$\square$

Consider the definite program $\mathcal{P}_2$ obtained from $\mathcal{P}_1$ by adding the clauses

$$\texttt{concat}_n(x, y, z) \leftarrow \texttt{conc}_n(c, x, c, y, c, z)$$

for all $1 < n \leq 2N$. The following lemma is an obvious consequence of Lemma 6.2:

Lemma 6.3 Let U, V, W be words on Σ. Then the atom $\texttt{concat}_n(cU, cV, cW)$ is true in the perfect model of $\mathcal{P}_2$ if and only if U, V have length 2^{n-1} and $UV = W$. $\square$

We call a *hypertile of rank n* any tiling of the square $2^n \times 2^n$. We shall now show how to represent hypertiles of rank n by definite programs of size polynomial in n.

To this end, we define predicates $\texttt{hypertile}_n(s, t)$ for $1 \leq n \leq N$ denoting that t is a word encoding a hypertile of rank n with the tile s in the top left corner.

A hypertile of rank 1 is a square of size 2×2

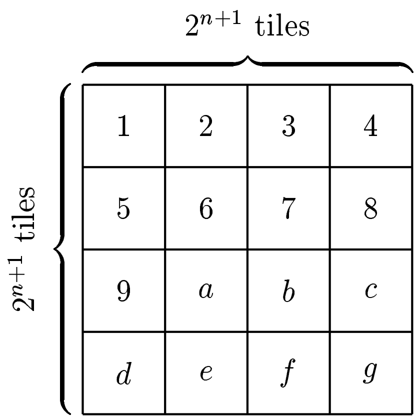
1	2
3	4

satisfying the compatibility conditions on tiles. Therefore we define the predicate $\texttt{hypertile}_1$ by the clause

$$\begin{aligned}
&\texttt{hypertile}_1(x_1, x) \leftarrow \\
&\quad \texttt{concat}_1(x_1, x_2, x_{12}), \\
&\quad \texttt{concat}_1(x_3, x_4, x_{34}), \\
&\quad \texttt{concat}_2(x_{12}, x_{34}, x),
\end{aligned}$$

$\texttt{on}(x_1, x_3)$,
$\texttt{on}(x_2, x_4)$,
$\texttt{to}(x_1, x_2)$,
$\texttt{to}(x_3, x_4)$.

For $n > 1$ we make the following observation. Consider any square of tiles of size $2^{n+1} \times 2^{n+1}$, where $n \geq 1$:



This square is a tiling of size 2^{n+1} if and only if the following nine subsquares are tilings of size 2^n:

1	2		2	3		3	4
5	6		6	7		7	8

5	6		6	7		7	8
9	a		a	b		b	c

9	a		a	b		b	c
d	e		e	f		f	g

In view of this observation, we define $\texttt{hypertile}_{n+1}$ by the clause (where each occurrence of _ denotes a unique fresh variable)

$$
\begin{aligned}
&\texttt{hypertile}_{n+1}(y, x) \leftarrow \\
&\quad \texttt{concat}_{2n+2}(x_{12345678}, x_{9abcdefg}, x), \\
&\quad \texttt{concat}_{2n+1}(x_{1234}, x_{5678}, x_{12345678}), \\
&\quad \texttt{concat}_{2n+1}(x_{9abc}, x_{defg}, x_{9abcdefg}), \\
&\quad \texttt{concat}_{2n}(x_{12}, x_{34}, x_{1234}), \\
&\quad \texttt{concat}_{2n}(x_{56}, x_{78}, x_{5678}), \\
&\quad \texttt{concat}_{2n}(x_{9a}, x_{bc}, x_{9abc}), \\
&\quad \texttt{concat}_{2n}(x_{de}, x_{fg}, x_{defg}), \\
&\quad \texttt{concat}_{2n-1}(x_1, x_2, x_{12}), \\
&\quad \texttt{concat}_{2n-1}(x_3, x_4, x_{34}), \\
&\quad \texttt{concat}_{2n-1}(x_5, x_6, x_{56}), \\
&\quad \texttt{concat}_{2n-1}(x_7, x_8, x_{78}), \\
&\quad \texttt{concat}_{2n-1}(x_9, x_a, x_{9a}), \\
&\quad \texttt{concat}_{2n-1}(x_b, x_c, x_{bc}), \\
&\quad \texttt{concat}_{2n-1}(x_d, x_e, x_{de}), \\
&\quad \texttt{concat}_{2n-1}(x_f, x_g, x_{fg}),
\end{aligned}
$$

$\mathtt{concat}_{2n-1}(x_2, x_3, x_{23}),$
$\mathtt{concat}_{2n-1}(x_6, x_7, x_{67}),$
$\mathtt{concat}_{2n-1}(x_a, x_b, x_{ab}),$
$\mathtt{concat}_{2n-1}(x_e, x_f, x_{ef}),$
$\mathtt{concat}_{2n}(x_{12}, x_{56}, x_{1256}),$
$\mathtt{concat}_{2n}(x_{23}, x_{67}, x_{2367}),$
$\mathtt{concat}_{2n}(x_{34}, x_{78}, x_{3478}),$
$\mathtt{concat}_{2n}(x_{56}, x_{9a}, x_{569a}),$
$\mathtt{concat}_{2n}(x_{67}, x_{ab}, x_{67ab}),$
$\mathtt{concat}_{2n}(x_{78}, x_{bc}, x_{78bc}),$
$\mathtt{concat}_{2n}(x_{9a}, x_{de}, x_{9ade}),$
$\mathtt{concat}_{2n}(x_{ab}, x_{ef}, x_{abef}),$
$\mathtt{concat}_{2n}(x_{bc}, x_{fg}, x_{bcfg}),$

$\mathtt{hypertile}_n(y, x_{1256}),$
$\mathtt{hypertile}_n(\_, x_{2367}),$
$\mathtt{hypertile}_n(\_, x_{3478}),$
$\mathtt{hypertile}_n(\_, x_{569a}),$
$\mathtt{hypertile}_n(\_, x_{67ab}),$
$\mathtt{hypertile}_n(\_, x_{78bc}),$
$\mathtt{hypertile}_n(\_, x_{9ade}),$
$\mathtt{hypertile}_n(\_, x_{abef}),$
$\mathtt{hypertile}_n(\_, x_{bcfg}).$

Let the program $\mathcal{P}_3$ be obtained from $\mathcal{P}_2$ by adding the definitions of the predicates $\mathtt{hypertile}_n$ for all $1 \leq n \leq N$. By the analysis of the construction of these definitions and using Lemma 6.3, we obtain

Lemma 6.4 The atom $\mathtt{hypertile}_n(s, t)$ is true in the perfect model of $\mathcal{P}_3$ if and only if t represents a tiling of size $2^n \times 2^n$ with the tile s in the top left corner.

Now we can prove Theorem 6.1. Inclusion in *NEXPTIME* follows from Theorem 5.1. *NEXPTIME*-hardness follows from Lemma 6.4, since the program $\mathcal{P}_3$ can be constructed in time polynomial in N and the tiling program is *NEXPTIME*-complete.

Our proof used a signature that contained symbols for all tiles. By using the standard encoding of arbitrary alphabets by a two-letter alphabet, we can restrict the signature by two unary symbols.

6.2 Normal programs

In this section we show that the complexity of the SUCCESS problem for nonrecursive logic programs with negation and function symbols of arity at most one is complete for a complexity class intermediate between *NEXPTIME* and *EXPSPACE*. The following key theorem is due to (Ferrante & Rackoff 1979, Chapters 4 and 9):

Theorem 6.5 $Th(TA(1,2,0))$ can be decided in $DSPACE(2^{O(n)})$ and is $NTIME(2^{O(n)})$-hard w.r.t. loglin reducibility[2]. □

(Volger 1983*b*, Volger 1983*a*) improved it[3] to

Theorem 6.6 $Th(TA(1,2,0))$ is $LATIME(2^{O(n)})$-complete. □

$LATIME(2^{O(n)})$ is a class of problems solvable by alternating Turing machines in time $2^{O(n)}$ with linear number of alternations[4]. By (Chandra, Kozen & Stockmeyer 1981), $LATIME(2^{O(n)}) \subseteq DSPACE(2^{O(n)})$. Also, obviously, $NTIME(2^{O(n)}) \subseteq LATIME(2^{O(n)})$. Both inclusions are *presumably* proper.

Theorem 6.6 and our Theorem 3.1 imply the following

Corollary 6.7 SUCCESS$(1,2,0)$ is $LATIME(2^{O(n)})$-complete for nonrecursive normal problems. □

The lower bound for the nonrecursive SUCCESS$(\geq 1, \geq 2, _)$ with negation also follows immediately:

Corollary 6.8 SUCCESS$(_, \geq 2, _)$ is $LATIME(2^{O(n)})$-hard for nonrecursive normal programs.

The upper bound appears to be of the same kind:

Theorem 6.9 SUCCESS$(_, \geq 2, 0)$ is in $LATIME(2^{O(n)})$ for nonrecursive logic programs with negation.

---

[2] Consequently, *NEXPTIME*-hard w.r.t. polynomial reducibility, cf., (Johnson 1990).

[3] At the time when (Ferrante & Rackoff 1979) was written, the complexity classes defined simultaneously in terms of time, space, and alternations were not yet well known. They first appeared in (Berman 1977, Bruss & Meyer 1980, Berman 1980).

[4] (Johnson 1990) calls this class $TA(2^{O(n)}, n)$, which clashes with our usage of $TA()$ for 'term algebras'

Proof. The proof of (Volger 1983*b*) is a straightforward corollary of the proof of (Ferrante & Rackoff 1979), and is based on the simple observation that $Th(TA(1,2,0))$ may be decided within $LATIME(2^{O(n)})$.

In fact, the $DSPACE(2^{O(n)})$ upper bound of (Ferrante & Rackoff 1979) is a consequence of their technically difficult result based on application of complexity-tailored Ehrenfeucht-Fraïssé games, which is as follows.

The *depth* of a term t, denoted $depth(t)$, is defined inductively as follows:

$$\begin{array}{lcl} depth(a) & = & 0, \text{ if } a \text{ is a constant} \\ depth(f(t_1,\ldots,t_n)) & = & 1+\max(depth(t_1),\ldots,depth(t_n)). \end{array}$$

Lemma 6.10 (Ferrante & Rackoff 1979) In $TA(1,2,0)$ a quantified prenex sentence is true if and only if the corresponding boundedly quantified sentence, where each quantifier runs over ground terms of depth *at most* exponential of the length of the sentence. □

From this it is immediate that a decision algorithm can be implemented in $LATIME(2^{O(n)})$. Indeed, it is clear that the brute-force test of the validity of a boundedly quantified sentence of $TA(1,2,0)$ can be performed by an alternating Turing machine within exponential time with linear number of alternations (corresponding to the alternations of quantifiers in the sentence).

The analysis of the proof in (Ferrante & Rackoff 1979) shows that it works not only for two successors, but also for arbitrary signatures containing symbols of arity at most one. Lemma 6.10 above generalizes to

Lemma 6.11 In $TA(\_,\geq 2,0)$ a quantified prenex sentence Φ is true if and only if the corresponding boundedly quantified sentence, where each quantifier runs over ground terms of a finite unary signature (whose size is linear in the size of Φ) of depth *at most* exponential of the length of Φ. □

This lemma gives us the $LATIME(2^{O(n)})$ upper bound for $Th(TA(\geq 1,\geq 2,0))$, and by Theorem 3.1, also for $\mathsf{SUCCESS}(\_,\geq 2,\_)$. □

Summarizing, the complexity of the $\mathsf{SUCCESS}$ problem in unary signatures is as follows:

Theorem 6.12 The $\mathsf{SUCCESS}(\_,\geq 2,\_)$ problem for nonrecursive logic programs with negation is $LATIME(2^{O(n)})$-complete. □

7 Range-restricted programs

We start with two general statements about range-restricted programs. The first one asserts that in the range-restricted case only symbols occurring in the program matter.

Lemma 7.1 Let $\mathcal{P}$ be a nonrecursive range-restricted program of signature Σ, P be a predicate different from equality, and $t_1, \ldots, t_n$ be ground terms of Σ such that $P(t_1, \ldots, t_n)$ is true in the perfect model of $\mathcal{P}$ in Σ. Then $t_1, \ldots, t_n$ are built of symbols occurring in $\mathcal{P}$. □

The proof of this theorem is straightforward.

Lemma 7.2 Let $\mathcal{P}$ be a nonrecursive range-restricted program in a signature Σ, P be a predicate different from equality, and $t_1, \ldots, t_n$ be ground terms of Σ such $P(t_1, \ldots, t_n)$ is true in the perfect model of $\mathcal{P}$ in Σ. Let K be the maximal number of occurrences of function symbols in clauses of $\mathcal{P}$, and N be the number or predicates defined in $\mathcal{P}$. Then $depth(t_i) \leq NK$ for all i.

Proof. We assume, without loss of generality, that the program defines predicates $P_0, \ldots, P_{N-1}$ and each P_{i+1} is defined in terms of P_i. We prove, by induction on i, that whenever $P_i(t_1, \ldots, t_m)$ is true in the perfect model, we have $depth(t_j) \leq (i+1)K$, for all j.

Indeed, let
$$C' = (P_{i+1}(t_1, \ldots, t_m) \leftarrow L_1, \ldots, L_k)$$
be any ground instance of a clause $C \in \mathcal{P}$. By induction hypothesis and using the fact that C is range-restricted, we get that all terms occurring as arguments in literals L_j have depth $\leq (i+1)K$.

Since every variable occurring in the head of C also occurs in the body, and the head of C contains at most K occurrences of function symbols, we conclude that any term in the head of C' has depth $\leq (i+2)K$. Since this is true for all clauses defining P_{i+1}, it proves the statement for $i+1$. □

The case of programs without function symbols was already considered in Section 4. We now proceed to range-restricted programs in the case of unary and then arbitrary function symbols.

7.1 Programs with unary function symbols

The definition of the predicates $\mathtt{conc}_n$ in Section 6 uses non-range-restricted clauses. Interestingly, the $\mathsf{SUCCESS}$ problem for monadic signatures and range-restricted clauses is essentially simpler, even in presence of negation.

Theorem 7.3 $\mathsf{SUCCESS}(\_, \geq 2, 0)$ is *PSPACE*-complete for nonrecursive range-restricted normal programs.

Proof. *PSPACE*-hardness follows from Theorem 4.4. In order to prove inclusion in *PSPACE*, we show how to decide the truth using an alternating polynomial time algorithm.

Let a program $\mathcal{P}$ of a signature $(\_, \geq 2, 0)$ define predicates $P_0, \ldots, P_N$ so that each predicate P_{i+1} is defined in terms of P_i.

We show, by induction on i, how to decide whether $P_i(t_1, \ldots, t_n)$ is true (or false) for ground terms $t_1, \ldots, t_n$. Let K be the maximal number of occurrences of function symbols in clauses of $\mathcal{P}$. By Lemma 7.2, we can assume that the depth of each t_i is bound by a polynomial in the size of $\mathcal{P}$. Find all clauses in $\mathcal{P}$ that have an instance

$$P_i(t_1, \ldots, t_n) \leftarrow A_1, \ldots, A_k, \neg B_1, \ldots, \neg B_m,$$

where A_j, B_l are atoms. By Lemmas 7.2 and 7.1, each atom A_j can only be true on ground terms of depth $\leq (i+2)K$ built from atoms occurring in $\mathcal{P}$. Since the signature is monadic, we can guess all such terms using nondeterministic OR-branching with polynomial time on each branch. After we guess such terms, all A_j become ground. Since the clause is range-restricted, all B_l also become ground. Using AND-branching, we check whether each $A_1, \ldots, A_k, \neg B_1, \ldots, \neg B_m$ is true in the perfect model of $\mathcal{P}$.

Note that the algorithm makes at most $2N$ alternations. $\square$

7.2 Arbitrary range-restricted programs

In the rest of this section we consider range-restricted programs with negation. The complexity of this case is characterized by the following theorem.

Theorem 7.4 $\mathsf{SUCCESS}(\_, \_, \geq 1)$ is *LATIME*$(2^{O(n)})$-complete for nonrecursive range-restricted programs. $\square$

Inclusion in $LATIME(2^{O(n)})$ uses the same proof as Theorem 7.3. The only difference is that when we guess ground terms of polynomial depth using nondeterministic OR-branching, we have to make an exponentially deep number of guesses. Thus, the algorithm runs in exponential time, but still with a linear number of alternations.

The proof $LATIME(2^{O(n)})$-hardness will use reduction from the theory of bounded concatenation discussed in the next section.

7.3 Boundedly quantified theory of bounded concatenation

We give the definition following (Bruss & Meyer 1980).

Fix a finite alphabet A with at least two symbols. Let $L(A)$ be the first-order language with equality, with constants a for each $a \in A$, and whose only atomic formulas (other than equalities) are of the form $bcat(x, y, z, \mathbf{n})$, where $\mathbf{n}$ is the unary numeral for the nonnegative integer n. Then for any function $t : N \to N$, we define *t-bounded concatenation theory* $t\text{-}BCT(A)$ as the set of true sentences in $L(A)$ under the following interpretation: the underlying domain is the set A^* of words over A, the constant symbols denote the elements $a \in A$, and for all words U, V, W, $bcat(U, V, W, \mathbf{n})$ is true if (i) U is the concatenation of V and W, and (ii) the length of U is at most $t(n)$.

Without loss of generality one may suppose that the alphabet A consists of two symbols, 0 and 1. (Volger 1983*b*) basing on (Ferrante & Rackoff 1979) proved $LATIME(2^{O(n)})$-completeness of $2^n\text{-}BCT(A)$.

Since formulas of $t\text{-}BCT(A)$ may contain equalities, one cannot directly claim that a sentence of $t\text{-}BCT(A)$ with arbitrarily quantified variables is equivalent to a boundedly quantified sentence. This is because the sizes of values of variables in equalities are unbounded. We need the following lemma allowing us to bound quantified variables in sentences of $2^n\text{-}BCT(A)$. For $Q \in \{\forall, \exists\}$ let $Qx \le 2^{cn}$ means that a quantified variable ranges over words of length at most 2^{cn}.

Lemma 7.5 In $2^n\text{-}BCT(A)$, given a quantified prenex sentence

$$F = Q_1 x_1 \dots Q_k x_k \; \Phi$$

of length n one can construct in polynomial time a boundedly quantified sentence

$$F_b = Q_1 x_1 \leq 2^{cn} \ldots Q_r x_r \leq 2^{cn}\ \Theta$$

true iff F is true.

Proof. First, we show that the theory of 2^n-bounded concatenation 2^n-$BCT(A)$ is polynomial time reducible to the first-order theory of two successors
$Th(\langle\{0,1\}^*, r_0, r_1\rangle)$ with $r_0(x) = x0$ and $r_1(x) = x1$. Second, we use once again Lemma 6.10 due to (Ferrante & Rackoff 1979), showing that any sentence of $Th(\langle\{0,1\}^*, r_0, r_1\rangle)$ is true iff the corresponding boundedly quantified sentence (with exponential bounds) is true. Third, we reduce boundedly quantified sentences of $Th(\langle\{0,1\}^*, r_0, r_1\rangle)$ to boundedly quantified sentences of 2^n-$BCT(A)$. This gives the desired conclusion.

Let $empty(u) = \neg\exists v(u \approx r_0(v) \vee u \approx r_1(v))$ be the formula of
$Th(\langle\{0,1\}^*, r_0, r_1\rangle)$ expressing that u has no predecessor, i.e., u is the empty word.

We need to write explicit definitions for the predicates $bcat(x, y, z, \mathbf{n})$ in $Th(\langle\{0,1\}^*, r_0, r_1\rangle)$. To do this we start by defining auxiliary predicates $bc(x, y, z, \mathbf{n})$ meaning that the length of y is at most 2^n and x is the concatenation of y and z:

$$\begin{aligned} bc(x, y, z, \mathbf{0}) &= \exists u(empty(u) \wedge ((x \approx r_0(z) \wedge y \approx r_0(u)) \vee \\ &\quad (x \approx r_1(z) \wedge y \approx r_1(u)) \vee (x \approx z \wedge y \approx u))), \\ bc(x, y, z, \mathbf{sn}) &= \exists y_1 y_2 z' \exists u(empty(u) \wedge bc(y_2, y_2, u, \mathbf{n}) \wedge \\ &\quad bc(y, y_1, y_2, \mathbf{n}) \wedge bc(z', y_2, z, \mathbf{n}) \wedge bc(x, y_1, z', \mathbf{n})). \end{aligned}$$

By Lemma 3.2, we can write an explicit definition for $bc(x, y, z, \mathbf{n})$ of size polynomial (even linear) in $\mathbf{n}$.

Using $bc(x, y, z, \mathbf{n})$, we can explicitly define $bcat(x, y, z, \mathbf{n})$ as follows:

$$bcat(x, y, z, \mathbf{n}) = \exists u(empty(u) \wedge bc(x, x, u, \mathbf{n}) \wedge bc(x, y, z, \mathbf{n})).$$

Now, given a prenex sentence

$$F = Q_1 x_1 \ldots Q_k x_k\ \Phi$$

of 2^n-$BCT(A)$ we can construct in polynomial time a prenex sentence

$$G = Q_1 x_1 \dots Q_k x_k Q_{k+1} x_{k+1} \dots Q_{k+l} x_{k+l} \Psi$$

of $Th(\langle\{0,1\}^*, r_0, r_1\rangle)$ true iff F is true. Note that additional quantifiers will result from *empty*, *bcat* and Lemma 3.2.

By Lemma 6.10, G is true in $Th(\langle\{0,1\}^*, r_0, r_1\rangle)$ iff for some $c > 0$ independent of n the boundedly quantified sentence

$$G_b = Q_1 x_1 \leq 2^{cn} \dots Q_k x_k \leq 2^{cn} Q_{k+1} x_{k+1} \leq 2^{cn} \dots Q_{k+l} x_{k+l} \leq 2^{cn}\ \Psi$$

is true in $Th(\langle\{0,1\}^*, r_0, r_1\rangle)$, where n is the length of sentence G.

We can now translate the sentence G_b of $Th(\langle\{0,1\}^*, r_0, r_1\rangle)$ back into a sentence F_b of 2^n-$BCT(A)$ by modifying the matrix Ψ of G_b as follows:

- adding three boundedly quantified variables $\exists e, u, v \leq 1$,
- adding conjunctively $bcat(e, e, e, \mathbf{0})$ (to mean that e is the empty word),
- adding conjunctively $bcat(u, u, e, \mathbf{0}) \wedge \neg bcat(u, u, u, \mathbf{0})$ (to mean that u is of length 1),
- adding conjunctively $bcat(v, v, e, \mathbf{0}) \wedge \neg bcat(v, v, v, \mathbf{0}) \wedge \neg bcat(u, v, e, \mathbf{0})$ (to mean that v is also of length 1, but different from u; thus $u = 0$ and $v = 1$, or vice versa, it does not matter for our purposes),
- replacing in Ψ each $y \approx r_0(x)$ with $bcat(y, u, x, \mathbf{cn})$ (since we know that all quantified variables are bounded in G_b),
- likewise, replacing in Ψ each $y \approx r_1(x)$ with $bcat(y, v, x, \mathbf{cn})$,
- likewise, replacing in Ψ each $y \approx x$ with $bcat(y, e, x, \mathbf{cn})$.

Clearly, the resulting boundedly quantified sentence F_b of 2^n-$BCT(A)$ is true iff G_b is true in $Th(\langle\{0,1\}^*, r_0, r_1\rangle)$.

We now have the following chain of reductions: given a sentence F of 2^n-$BCT(A)$ we constructed in polynomial time the boundedly quantified sentence F_b of 2^n-$BCT(A)$ true in 2^n-$BCT(A)$ if F is true, as needed. $\square$

7.4 Reduction of the theory of bounded concatenation

In this section we complete the proof of Theorem 7.4. To prove *LATIME*($2^{O(n)}$)-hardness, we reduce 2^n-$BCT(A)$ to the SUCCESS problem for nonrecursive range-restricted programs in any signature Σ with a function symbol of arity ≥ 2. For simplicity, we assume that the signature Σ consists of three constants $0, 1, \varepsilon$ and a binary function symbol f.

First, we choose a representation of nonempty words on $\{0,1\}$ by binary trees (or terms) constructed using $f, 0, 1$. For any such term t we define the corresponding word $w(t)$ as follows:

$$\begin{array}{lcl} w(0) & = & 0 \\ w(1) & = & 1 \\ w(f(s,t)) & = & w(s)w(t) \end{array}$$

Note that words do not have a unique representation.

Let us fix some positive integer N. Consider the following nonrecursive clauses defining predicates $\texttt{nonempty\_word}_n$:

$$\texttt{nonempty\_word}_0(0) \leftarrow$$
$$\texttt{nonempty\_word}_0(1) \leftarrow$$

and the clauses

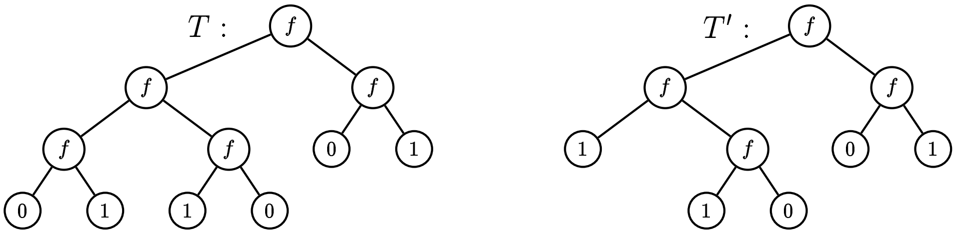
$$\texttt{nonempty\_word}_{n+1}(f(x,y)) \leftarrow \texttt{nonempty\_word}_n(x), \texttt{nonempty\_word}_n(y)$$
$$\texttt{nonempty\_word}_{n+1}(x) \leftarrow \texttt{nonempty\_word}_n(x)$$

for all $n < N$. Evidently, we have $\texttt{nonempty\_word}_n(t)$ if and only if t is a ground term of depth $\leq n$ built from $f, 0, 1$, encoding a nonempty word.

We recursively define the notion *a tree T' is obtained from a tree T by removing the leftmost leaf.* This holds if

1. $T = f(a,t)$, a is a constant and $T' = t$;
2. $T = f(t_1, t_2)$, t_1 is not a constant, t_1' is obtained from t_1 by removing the leftmost leaf and $T' = f(t_1', t_2)$.

The removal of the leftmost leaf is illustrated by the following picture.



Now we define two other series of predicates:

- $\mathtt{weq}_n(x, y)$ for all $n \leq N$ mean that x, y are terms of depth $\leq n$ and $w(x) = w(y)$;

- $\mathtt{diff}_n(x, y, z)$ for all $n \leq N$ mean that x, y, z are terms of depth $\leq n$, $w(y)w(z) = w(x)$ and z is obtained from x by a sequence of removals of leftmost leaves.

The definition of $\mathtt{weq}_0$ is obvious:

$\mathtt{weq}_0(0, 0) \leftarrow;$
$\mathtt{weq}_0(1, 1) \leftarrow.$

The definition of $\mathtt{diff}_0$ is empty, because $\mathtt{diff}_0(x, y, z)$ is never true.

In order to define $\mathtt{weq}_{n+1}$ for trees $f(x, y)$ and $f(u, v)$ we should consider three possible cases: (i) $w(x) = w(u)$; (ii) $w(x)$ is a prefix of $w(u)$; and (iii) $w(u)$ is a prefix of $w(x)$. This gives rise to the following three clauses:

$\mathtt{weq}_{n+1}(f(x, y), f(u, v)) \leftarrow \mathtt{weq}_n(x, y), \mathtt{weq}_n(u, v);$
$\mathtt{weq}_{n+1}(f(x, y), f(u, v)) \leftarrow \mathtt{diff}_n(u, x, z), \mathtt{diff}_n(y, z, v'), \mathtt{weq}_n(v', v);$
$\mathtt{weq}_{n+1}(f(x, y), f(u, v)) \leftarrow \mathtt{diff}_n(x, u, z), \mathtt{diff}_n(v, z, y'), \mathtt{weq}_n(y', y).$

Finally, we add a self-explaining clause:

$\mathtt{weq}_{n+1}(x, y) \leftarrow \mathtt{weq}_n(x, y).$

In order to define $\mathtt{diff}_{n+1}$ for trees $f(x, y)$ and $f(u, v)$ we should consider five possible cases: (i) $w(x)$ is a prefix of $w(u)$; (ii) $w(x) = w(u)$; (iii) $w(u)$ is a prefix of $w(x)$ and $w(x)$ is a prefix of $w(u)w(v)$; (iv) $w(x) = w(u)w(v)$; and (v) $w(u)w(v)$ is a prefix of $w(x)$. This gives rise to the following five clauses.

$\mathtt{diff}_{n+1}(f(x,y), f(u,v), w) \leftarrow \mathtt{diff}_n(u,x,z), \mathtt{diff}_n(y,z,w);$
$\mathtt{diff}_{n+1}(f(x,y), f(u,v), w) \leftarrow \mathtt{weq}_n(x,u), \mathtt{diff}_n(y,v,w);$
$\mathtt{diff}_{n+1}(f(x,y), f(u,v), w) \leftarrow \mathtt{diff}_n(x,u,z_1), \mathtt{diff}_n(v,z_1,z_2), \mathtt{diff}_n(y,z_2,w);$
$\mathtt{diff}_{n+1}(f(x,y), u, y) \leftarrow \mathtt{weq}_n(x,u), \mathtt{nonempty\_word}_n(y);$
$\mathtt{diff}_{n+1}(f(x,y), u, f(z,y)) \leftarrow \mathtt{diff}_n(x,u,z), \mathtt{nonempty\_word}_n(y).$

Note that the last two clauses also cover the case when the second argument of $\mathtt{diff}_n$ is a constant. As before, we have to add

$\mathtt{diff}_{n+1}(x,y,z) \leftarrow \mathtt{diff}_n(x,y,z).$

Before, we only encoded nonempty words by terms using $f, 0, 1$. Now we also assume that ε encodes the empty word, i.e. $w(\varepsilon)$ is the empty word, and define the 2^n-bounded concatenation:

- for each $n \leq N$, $\mathtt{conc}_n(x,y,z)$ means that x, y, z are terms of depth $\leq n$ and $w(x)w(y) = w(z)$.

First, we define predicates $\mathtt{word}_n$ defining terms of depth $\leq n$ denoting (maybe empty) words:

$\mathtt{word}_n(\varepsilon) \leftarrow;$
$\mathtt{word}_n(x) \leftarrow \mathtt{nonempty\_word}_n(x);$

Then we use the clauses

$\mathtt{conc}_n(\varepsilon, x, x) \leftarrow \mathtt{word}_n(x);$
$\mathtt{conc}_n(x, \varepsilon, x) \leftarrow \mathtt{word}_n(x);$
$\mathtt{conc}_n(x,y,z) \leftarrow \mathtt{diff}_n(z,x,y'), \mathtt{weq}_n(y', y).$

Note that all these clauses are nonrecursive, range-restricted, and can be constructed in time polynomial in N.

Now we show how to reduce the theory of bounded concatenation to the SUCCESS problem for nonrecursive range-restricted clauses.

By Lemma 7.5 we can assume that all quantifiers in formulas of 2^n-$BCT(A)$ are bounded by 2^{cn}. Let φ be any formula of size $\leq n$ whose free variables are $\bar{x}$. Define N as cn and build the definition of $\mathtt{conc}_m$ for all $m \leq N$ as above. By induction on φ we construct a nonrecursive logic program $\mathcal{P}$ defining a predicate $P_\varphi(\bar{x})$ with the following property.

Lemma 7.6 For all ground terms $t_1, \ldots, t_n$ of depth $\leq n$, the formula $P(t_1, \ldots, t_n)$ is true in the perfect model of $\mathcal{P}$ if and only if $\varphi(w(t_1), \ldots, w(t_n))$ is true in 2^n-$BCT(A)$.

We can assume that φ is constructed using only $\neg, \wedge$, and $\exists$. If φ is an atomic formula $bcat(x, y, z, \mathbf{n})$, we define P_φ as follows:

$$P_\varphi(x, y, z) \leftarrow \mathtt{conc}_n(y, z, x)$$

The cases of conjunction and existential quantifier are obvious:

$$
\begin{array}{l}
P_{\varphi_1 \wedge \varphi_2}(\bar{x}) \leftarrow \\
\qquad P_{\varphi_1}(\bar{x}_1), \\
\qquad P_{\varphi_2}(\bar{x}_2). \\
P_{\exists y \leq 2^{cn} \varphi}(\bar{x}) \leftarrow \\
\qquad P_\varphi(\bar{x}, y), \\
\qquad \mathtt{word}_N(y)
\end{array}
$$

Finally, the negation is handled in the following way:

$$
\begin{array}{l}
P_{\neg\varphi}(x_1, \ldots, x_m) \leftarrow \\
\qquad \neg P_\varphi(x_1, \ldots, x_m), \\
\qquad \mathtt{word}_N(x_1), \\
\qquad \ldots, \\
\qquad \mathtt{word}_N(x_m).
\end{array}
$$

The proof of Lemma 7.6 is straightforward. Note that the program defining P_φ is range-restricted, nonrecursive and can be constructed in time polynomial in the size of φ. Thus, 2^n-$BCT(A)$ is polynomial time reducible to the SUCCESS problem for nonrecursive range-restricted normal programs which completes the proof of Theorem 7.4. $\square$

8 Other kinds of complex values

In this section we briefly consider various formalizations of finite sets and multisets. The complexity of the corresponding nonrecursive fragment of logic programming with complex values depends on this formalization and the signature, i.e., the set of operations available on the values. We consider

1. *colored finite sets* (Dovier et al. 1996);
2. *untyped finite sets* (Dantsin & Voronkov 1997*a*, Dantsin & Voronkov 1997*b*);
3. *typed finite sets* (Abiteboul & Beeri 1995).

The definitions and subsequent results can be straightforwardly generalized to finite multisets.

8.1 Formalizations of finite sets

Universe with colored sets. In order to construct the *domain of colored sets* (Dovier et al. 1996), we need to add to a signature Σ a binary function symbol $\{\ldots \mid \ldots\}$, called the *set constructor*. A *color* is any term whose top function symbol is not $\{\,|\,\}$. A term $\{s_1 \mid \ldots \{s_n \,|\, t\}\}$, where t is a color, represents the *colored set* with elements $s_1, \ldots, s_n$ and the color t. Two colored sets are equal when they have the same elements and the same color. To represent the empty set, any color can be used. Note that a color may be a complex term whose subterms can be colored sets.

Universe with untyped sets. Untyped sets (Dantsin & Voronkov 1997*a*, Dantsin & Voronkov 1997*b*) are defined similarly to colored sets, but the only color is the constant $\emptyset$, representing the empty set. In order to formalize untyped sets, a two-sorted signature must be used. (In fact, colored sets were introduced in order to have completely unsorted language.)

Universe with typed sets. We shall present a typed universe for complex values containing finite sets following (Abiteboul & Beeri 1995).

We assume a set of *domain names* $\widehat{D_1}, \widehat{D_2}, \ldots$ and an infinite set of *attributes* $A_1, A_2, \ldots$ The domain names are associated with *domains* $D_1, D_2, \ldots$, i.e., non-empty sets whose elements are called *atomic values*.

Complex values are constructed from atomic ones using constructors. A type is associated with each value.

Definition 8.1 *Types* and *values* for this domain are defined as follows:

1. If $\widehat{D}$ is a domain name, then $\widehat{D}$ is an *atomic* type. For each $a \in D$, the element a is a value of this type.

2. If $T_1, \ldots, T_n$, where $n \geq 0$, are types and $A_1, \ldots, A_n$ are distinct attributes, then $[A_1 : T_1, \ldots, A_n : T_n]$ is a *tuple* type. If $v_1, \ldots, v_n$ are values of types $T_1, \ldots, T_n$, respectively, then $[A_1 : v_1, \ldots, A_n : v_n]$ is a value of the type.

3. If T is a type, then $\{T\}$ is a set type. Any finite set of values of type T is a value of type $\{T\}$.

Syntactic representation of values from the typed domain is similar to that for the untyped domain (in particular, we have the set constructor), but with the following changes.

1. Since the atomic domains are typed, the corresponding constants of the language are also typed.

2. We have *tuple terms* $[A_1 : t_1, \ldots, A_n : t_n]$ made of terms $t_1, \ldots, t_n$ with the natural interpretation.

3. The constants for the empty set and the set constructor satisfy the natural restrictions on their types.

In order to define semantics of nonrecursive logic programs over these universes, we need only one change compared to page 7. Instead of considering nonrecursive logic programs as sets of explicit definitions over the term algebra, we consider them over the corresponding universe.

(Dantsin & Voronkov 1997*a*, Dantsin & Voronkov 1997*b*) proved

Theorem 8.2 The SUCCESS problem for definite programs is *NEXPTIME*-complete for the following domains:

1. universe with colored sets and/or bags;

2. universe with untyped sets and/or bags;

3. universe with typed sets and/or bags.

The same holds for range-restricted definite programs. □

In the remaining part of the paper we address the complexity of nonrecursive logic programs over the three domains defined in this section.

8.2 Typed universe

Evidently, universes with typed and untyped sets correspond to two versions, typed and untyped, of the domain of hereditarily finite sets, see e.g., (Barwise 1975). In this section we settle complexity bounds for the typed universe. We need some background material on the theory of typed hereditarily finite sets.

Type theory Ω is a rudimentary fragment of L. Henkin's theory of propositional types (Henkin 1963). The language of Ω is a language of set theory, where every variable has a natural number type (written as a binary superscript) and there are two constants $\mathbf{0}$, $\mathbf{1}$ of type 0. The theory has a countable number of predicate symbols $\in^n$, for every natural number n. The interpretation of Ω is as follows: $\mathbf{0}$ denotes 0, $\mathbf{1}$ denotes 1, elements of type 0 are 0 and 1 and elements of types $n+1$ are sets of elements of type n.

For complexity considerations let us fix any reasonable encoding of formulas of Ω as binary strings and agree that a variable of Ω be represented by its type and its identification number within a type, both written in binary.

The validity problem for Ω is trivially decidable, because every quantifier runs over a *finite* domain, but *is not elementary recursive.* Even stronger, (Vorobyov 1997):

Theorem 8.3 Any Turing machine deciding Ω makes a number of steps exceeding

$$\exp_\infty(2^{c|\Phi|}) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\} \text{ height } 2^{c|\Phi|}$$

for some constant $c > 0$ and infinitely many sentences Φ of Ω. □

This improves the previously known lower bound (Meyer 1974, pp. 478–479), which was a logarithmically growing tower of twos.

Reduction of Ω to nonrecursive logic programs over the typed universe of sets. Here we apply the strong lower bound from Theorem 8.3 by polynomially reducing Ω to the SUCCESS problem for the typed universe. It follows that the latter problem has the same lower complexity bound.

We consider the following typed universe $\mathcal{S}$ of finite sets. There is only one atomic type with constants 0 and 1. We construct all set types as in Section 8.1, i.e., using constants for empty sets and set constructors for all set types.

For a sentence Φ of the theory Ω its polynomial-time translation to the SUCCESS problem for nonrecursive logic programs over $\mathcal{S}$ is defined exactly as in the proof of Theorem 3.1, except for the case of atomic formulas. The membership predicates $\in^n$ can be defined by the following clause:

$$\mathtt{member}_n(x, \{x \mid y\}) \leftarrow .$$

Henceforth, Theorem 8.3 yields

Theorem 8.4 The SUCCESS problem for nonrecursive logic programs over $\mathcal{S}$ is not elementary recursive. Even stronger: any algorithm solving the problem should necessarily spend time exceeding

$$\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\} \text{ height } 2^{c|\mathcal{P}|}$$

for some constant $c > 0$ and infinitely many instances $\mathcal{P}$ of the problem. $\square$

It is not difficult to show that the SUCCESS problem is decidable, and the upper bound is of the same kind. Summarizing, we get

Theorem 8.5 The SUCCESS problem for nonrecursive logic programs over $\mathcal{S}$ is in the complexity class *NONELEMENTARY*(2^n). $\square$

To our knowledge, this is a most complicated (in complexity-theoretic sense) of all decidable problems for which any upper bound is currently known.

It is easy to see that the complexity remains the same for any typed universe, where atomic domains contain a finite number of elements, since in this case all types will still have a finite number of elements.

Consider now the typed universe $\mathcal{S}'$ defined as $\mathcal{S}$, but with an infinite atomic domain. In this case we have the following fact.

Theorem 8.6 The $\mathsf{SUCCESS}$ problem for nonrecursive logic programs over $\mathcal{S}'$ is undecidable. $\square$

We leave a detailed proof to the reader. The idea is that it is possible to interpret the theory of a finite binary relation (Kalmar-Traktenbrot-Vaught-Rabin), cf., (Vaught 1960, Rabin 1964). Indeed, using the infinite atomic domain D we can express a finite set of any size (as any finite set S of elements of D) and interpret an arbitrary binary relation as a set of pairs $\{a, b\}$ with elements in D.

Analogously, if the atomic domain is finite, but variables and the membership predicate are untyped (polymorphic or interpreted over arbitrary strata) the theory and the problem are also undecidable.

8.3 Untyped universe

Similarly, we obtain

Theorem 8.7 The $\mathsf{SUCCESS}$ problems for nonrecursive logic programs over the untyped universe and the colored universe are undecidable. $\square$

This follows from the undecidability of the theory of a binary relation (Kalmar, Traktenbrot, Vaught, Rabin), cf., (Vaught 1960, Rabin 1964), or from Gandy's theorem (Barwise 1975).

9 Summary of results

The complexity of the success problem for nonrecursive logic programs over trees are summarized in the following table. In all cases we have completeness in the corresponding complexity class, except for *NONELEMENTARY* (in this case both lower and upper bounds are linearly growing towers of twos).

function symbols	no	unary	any
	not range-restricted		
no negation	*PSPACE*	*PSPACE*	*NEXPTIME*
with negation	*PSPACE*	*LATIME*($2^{O(n)}$)	*NONELEMENTARY*(n)
	range-restricted		
no negation	*PSPACE*	*PSPACE*	*NEXPTIME*
with negation	*PSPACE*	*PSPACE*	*LATIME*($2^{O(n)}$)

We briefly discuss the obtained complexity results, by comparing results in all signatures with the simplest class: definite range-restricted programs. In the case without function symbols (corresponding to nonrecursive datalog) the complexity does not change when we add negation and remove range-restriction. For signatures with unary function symbols, the complexity is the same as for signatures without function symbols, except for the class without range restriction and with negation. For arbitrary signatures, we have a "small" increase in complexity when we either remove range restriction or add negation. However if we do both, the complexity becomes nonelementary. In all cases except datalog, it is the combination of negation and non-range restrictedness that gives a big jump in complexity.

We also proved some complexity results about nonrecursive logic programs over sets. However the complexity of such programs over sets should be further investigated. For example, it is not known what is the complexity of the range-restricted fragment. Another interesting question is the complexity for programs with a fixed finite number of negations, both with and without range-restriction.

References

Abiteboul, S. & Beeri, C. (1995), 'The power of languages for the manipulation of complex values', *VLDB Journal* **4**, 727–794.

Abiteboul, S., Hull, R. & Vianu, V. (1995), *Foundation of Databases*, Addison-Wesley Publishing Co.

Andréka & Németi (1978), 'A generalized completeness of Horn clause logic seen as a programming language', *Acta Cybernetica* **4**, 3–10.

Apt, K. (1990), Logic programming, *in* J. Van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, Elsevier Science, Amsterdam, chapter 10, pp. 493–574.

Apt, K. & Blair, H. (1988*a*), Arithmetic classification of perfect models of stratified programs, *in* R. Kowalski & K. Bouwen, eds, 'Proceedings of the Fifth Joint International Conference and Symposium on Logic Programming (JICSLP-88)', The MIT Press, pp. 766–779.

Apt, K. & Blair, H. (1988*b*), Towards a theory of declarative knowledge, *in* J. Minker, ed., 'Foundations of Deductive Databases and Logic Programming', Morgan Kaufmann, pp. 89–148.

Barwise, J. (1975), *Admissible Sets and Structures*, Springer Verlag.

Beeri, C., Naqvi, S., Schmueli, O. & Tsur, S. (1991), 'Set constructors in a logic database language', *Journal of Logic Programming* **10**, 181–232.

Benedikt, M. & 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.

Berman, L. (1977), Pecise bounds for Presburger arithmetic and the reals with addition: preliminary report, *in* 'International Conference of Foundations of Computer Science', pp. 95–99.

Berman, L. (1980), 'The complexity of logical theories', *Theoretical Computer Science* **11**, 71–77.

Blair, H., Marek, V. & Schlipf, J. (1995), 'The expressiveness of locally stratified programs', *Annals of Mathematics and Artificial Intelligence* **15**(3/4), 209–229.

Bruss, A. R. & Meyer, A. R. (1980), 'On time-space classes and their relation to the theory of real addition', *Theoretical Computer Science* **11**, 59–69.

Chandra, A., Kozen, D. & Stockmeyer, L. (1981), 'Alternation', *Journal of the Association for Computing Machinery* **28**, 114–133.

Clark, K. (1978), Negation as failure, *in* H. Gallaire & J. Minker, eds, 'Logic and Data Base', Plenum Press, New York, pp. 293–322.

Dantsin, E. (1986), The complexity of Prolog without loops (in Russian), *in* 'Proceedings of the 4th Soviet Conference "Application of the Mathematical Logic Methods"', Vol. 2, Tallinn, pp. 112–113.

Dantsin, E., Eiter, T., Gottlob, G. & Voronkov, A. (1997), Complexity and expressive power of logic programming, *in* 'Proceedings Twelfth Annual IEEE Conference on Computational Complexity', Ulm, Germany, pp. 82–101.

Dantsin, E. & Voronkov, A. (1997*a*), Bag and set unification, UPMAIL Technical Report 150, Uppsala University, Computing Science Department.

Dantsin, E. & Voronkov, A. (1997*b*), Complexity of query answering in logic databases with complex values, *in* S. Adian & A. Nerode, eds, 'Logical Foundations of Computer Science. 4th International Symposium, LFCS'97', Vol. 1234 of *Lecture Notes in Computer Science*, Yaroslavl, Russia, pp. 56–66.

Dantsin, E. & Voronkov, A. (1997*c*), Complexity of query answering in logic databases with complex data, UPMAIL Technical Report 149, Uppsala University, Computing Science Department.

Dovier, A., Omodeo, E., Pontelli, E. & Rossi, G. (1996), '{log}: A language for programming in logic with finite sets', *Journal of Logic Programming* **28**(1), 1–44.

Falashi, M., Levi, G., Martelli, M. & Palamidessi, C. (1989), 'Declarative modeling of the operational behavior of logic languages', *Theoretical Computer Science* **69**(3), 289–318.

Ferrante, J. & Rackoff, C. W. (1979), *The computational complexity of logical theories*, Vol. 718 of *Lecture Notes in Mathematics*, Springer-Verlag.

Gallier, J. & Raatz, S. (1989), 'Extending SLD-resolution to equational Horn clauses using E-unification', *Journal of Logic Programming* **6**(3), 3–44.

Henkin, L. (1963), 'A theory of propositional types', *Fundamenta Mathematicae* **52**, 323–344.

Hodges, W. (1993), *Model theory*, Cambridge University Press.

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.

Johnson, D. S. (1990), A catalog of complexity classes, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. A, Elsevier Science, chapter 8, pp. 67–161.

Kanellakis, P., Kuper, G. & Revesz, P. (1995), 'Constraint query languages', *Journal of Computer and System Sciences* **51**, 26–52.

Kunen, K. (1987*a*), Answer sets and negation as failure, *in* '4th International Conference on Logic Programming', Vol. 1, The MIT Press, pp. 219–228.

Kunen, K. (1987*b*), 'Negation in logic programming', *Journal of Logic Programming* **4**, 289–308.

Kuper, G. (1990), 'Logic programming with sets', *Journal of Computer and System Sciences* **41**, 44–64.

Lifschitz, V. (1988), On the declarative semantics of logic programs with negation, *in* J. Minker, ed., 'Deductive Databases and Logic Programming', Morgan Kaufmann.

Lloyd, J. (1987), *Foundations of Logic Programming (2nd edition)*, 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.

Maher, M. (1992), A CLP view of logic programming, *in* 'Proc. Conf. on Algebraic and Logic Programming', Vol. 632 of *Lecture Notes in Computer Science*, pp. 364–383.

Maher, M. (1993), A logic programming view of CLP, *in* 'International Conference on Logic Programming', pp. 737–753.

Maĺcev, A. (1961*a*), Axiomatizable classes of locally free algebras of various types, *in* B. Wells III, ed., 'The Metamathematics of Algebraic Systems. Anatoliĭ Ivanovič Maĺcev. Collected Papers: 1936–1967', Vol. 66, North Holland, chapter 23, pp. 262–281.

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

Meyer, A. (1974), The inherent computational complexity of theories of ordered sets, *in* 'International Congress of Mathematicians', pp. 477–482.

Papadimitriou, C. (1994), *Computational Complexity*, Addison-Wesley.

Papadimitriou, C. & Yannakakis, M. (1997), On the complexity of database queries, *in* 'ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems', pp. 12–19.

Przymusinski (1988), On the declarative semantics of deductive databases and logic programs, *in* J. Minker, ed., 'Foundations of Deductive Databases and Logic Programming', Morgan Kaufmann, pp. 193–216.

Rabin, M. O. (1964), A simple method for undecidability proofs and some applications, *in* Y. Bar-Hillel, ed., 'Proc. Intern. Congress on Logic, Methodology and Philosophy of Science', Studies in Logic and the Foundations of Mathematics, pp. 58–68.

Schlipf, J. (1995), 'Complexity and undecidability results in logic programming', *Annals of Mathematics and Artificial Intelligence* **15**(3/4), 257–288.

Schmueli, O., Tsur, S. & Zaniolo, C. (1992), 'Compilation of set terms in the logic data language (LDL)', *Journal of Logic Programming* **12**, 89–119.

Stockmeyer, L. J. (1977), 'The polynomial-time hierarchy', *Theoretical Computer Science* **3**, 1–22.

Stockmeyer, L. & Meyer, A. (1973), Word problems requiring exponential time: Preliminary report, *in* 'Proc. ACM STOC', pp. 1–9.

Tärnlund, S.-A. (1977), 'Horn clause computability', *BIT* **17**, 215–216.

Van Gelder, A. (1988), Negation as failure using tight derivations for general logic programs, *in* J. Minker, ed., 'Deductive Databases and Logic Programming', Morgan Kaufmann, pp. 149–177.

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

Vardi, M. (1995), On the complexity of bounded-variable queries, *in* 'ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems', pp. 266–276.

Vaught, R. L. (1960), 'Sentences true in all constructive models', *Journal of Symbolic Logic* **25**(1), 39–53.

Volger, H. (1983*a*), 'A new hierarchy of elementary recursive decision problems', *Methods of Operations Research* **45**, 509–519.

Volger, H. (1983*b*), 'Turing machines with linear alternation, theories of bounded concatenation and the decision problem of first order theories (Note)', *Theoretical Computer Science* **23**, 333–337.

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

Vorobyov, S. (1997), The "hardest" natural decidable theory, *in* 'Proc. IEEE Conference on Logic in Computer Science (LICS)', pp. 294–305.

I N F O R M A T I K

Below you find a list of the most recent technical reports of the research group *Logic of Programming* at the Max-Planck-Institut für Informatik. They are available by anonymous ftp from our ftp server `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Birgit Hofmann
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-97-2-009	A. Bockmayr, F. Eisenbrand	On the Chvátal Rank of Polytopes in the 0/1 Cube
MPI-I-97-2-008	A. Bockmayr, T. Kasper	A Unifying Framework for Integer and Finite Domain Constraint Programming
MPI-I-97-2-007	P. Blackburn, M. Tzakova	Two Hybrid Logics
MPI-I-97-2-006	S. Vorobyov	Third-order matching in $\lambda\rightarrow$-Curry is undecidable
MPI-I-97-2-005	L. Bachmair, H. Ganzinger	A Theory of Resolution
MPI-I-97-2-003	U. Hustadt, R.A. Schmidt	On evaluating decision procedures for modal logic
MPI-I-97-2-002	R.A. Schmidt	Resolution is a decision procedure for many propositional modal logics
MPI-I-97-2-001	D.A. Basin, S. Matthews, L. Viganò	Labelled modal logics: quantifiers
MPI-I-96-2-010	A. Nonnengart	Strong Skolemization
MPI-I-96-2-009	D.A. Basin, N. Klarlund	Beyond the Finite in Automatic Hardware Verification
MPI-I-96-2-008	S. Vorobyov	On the decision complexity of the bounded theories of trees
MPI-I-96-2-007	A. Herzig	SCAN and Systems of Conditional Logic
MPI-I-96-2-006	D.A. Basin, S. Matthews, L. Viganò	Natural Deduction for Non-Classical Logics
MPI-I-96-2-005	A. Nonnengart	Auxiliary Modal Operators and the Characterization of Modal Frames
MPI-I-96-2-004	G. Struth	Non-Symmetric Rewriting
MPI-I-96-2-003	H. Baumeister	Using Algebraic Specification Languages for Model-Oriented Specifications
MPI-I-96-2-002	D.A. Basin, S. Matthews, L. Viganò	Labelled Propositional Modal Logics: Theory and Practice
MPI-I-96-2-001	H. Ganzinger, U. Waldmann	Theorem Proving in Cancellative Abelian Monoids
MPI-I-95-2-011	P. Barth, A. Bockmayr	Modelling Mixed-Integer Optimisation Problems in Constraint Logic Programming