数理解析研究所講究録
第 790 巻 1992 年 269-275

Logspace Bounded Alternation and Logical Query Programs

Yasuo OKABE (岡部 寿男)* and Takao TSUDA (津田 孝夫)

Department of Information Science, Kyoto University

Abstract

We argue on the computational complexity of function-free Horn clause query programs, called logical query programs. We show a close relation ship between logical query programs and logspace bounded alternating Turing machines. First we present an alternating logspace algorithm for computing a logical query program. The "fringe complexity" of the program corresponds to the tree-size complexity of the ATM. Next we give a logical query program which simulates logspace bounded alternation. We also show that linear logical query programs are closely related to logspace bounded nondeterministic Turing machines. As a result, well-known complexity classes such as $\mathcal{NL}$, $LOGCFL$, $\mathcal{NC}$, and $\mathcal{P}$ are characterized via logical query programs. A main result is that the basic theorem problem of linear logical query programs is $\mathcal{NL}$-complete. This suggests that many results of complexity theory can be translated in terms of logic programs. As an example, we show that the negation o a linear program is also linear, by applying Immerman's theorem.

1 Introduction

Logic programs have been attracting considerable attention as languages for "fifth generation computers". Programs without function symbols, called logical query programs or Datalog programs, are regarded as particularly important because of its simplicity. They are also utilized as a query language for deductive databases. In this paper, we investigate theoretically the computational complexity of logical query programs, and closely relate it to those of conventional computation models, especially, that of alternating Turing machines.

Shapiro [8] first showed that there is a close relationship between logic programs and alternating Turing machines (ATMs) [2]. He regarded a logic program (with function symbols) as an ATM, which is given a initial goal as an input string. He introduced three complexity measures for logic programs, i.e., depth complexity, goal-size complexity, and length complexity, and showed that these three correspond to, respectively, time complexity, space complexity, and tree-size complexity [6], of ATMs. Okabe et al. [5] generalized the relation to ATMs with sublinear time and space complexity (see also [9]). They show a hierarchy of well-known complexity classes of problems with respect to the complexity of logic programs. These results are, however, not applicable to logic query programs directly, since Shapiro's (or even Okabe-et-al.'s) simulation cannot distinguish IDB (Intensional Database) rules and EDB (Extensional Database) facts. An EDB instance of logical query programs must be regarded rather as an input than as a part of a program.

Formalization of logic query programs as a theoretical model of computation was given by Ullman et al. [11][12]. They adopted the size of "fringes" of a derivation as the complexity measure, showed a PRAM algorithm which computes the interpretation of a logic query program, and showed that programs with the polynomial fringe property is in $\mathcal{NC}$, i.e., can be computed in time $(\log n)^{O(1)}$ by a PRAM. They also show that some logic query programs are logspace complete for $\mathcal{P}$ ($\mathcal{P}$-complete).

We here show a new characterization of logical query programs as logspace bounded alternating Turing machines. First we present an alternating logspace algorithm for computing a logical query program. The fringe complexity of the program is related to the tree-size complexity of an ATM. Next we give a logical query program which simulates any logspace bounded ATM. From the fact that $\mathcal{P} = ALOGSPACE$, we have got another proof of the $\mathcal{P}$-completeness.

We also show that linear logical query programs are closely related to logspace bounded nondeterministic Turing machines. This suggests us the limit of the power of linear recursions. As a corollary, it is proved that linear logical query program is logspace complete for $\mathcal{NL}$ ($NLOGSPACE$)

Well-known complexity classes such as $\mathcal{NL}$, $LOGCFL$, $\mathcal{NC}$, and $\mathcal{P}$ are characterized via logical query programs. The class of problems computable by logical query programs is just $\mathcal{P}$ [11], while general logic programs with function symbols have as much power as that of Turing machines. We can now say that $LOGCFL$, viz., the class of languages logspace reducible to context free languages (CFL), is equivalent to the class of problems computable by logical query programs with the polynomial fringe property. This is a new characterization of CFL via logic programs. Similarly, $\mathcal{NC}$ is the class of programs with the superpolynomial fringe property, and $\mathcal{NL}$ is the class of linear programs.

*E-mail: okabe@kuis.kyoto-u.ac.jp

Many results of complexity theory obtained on conventional models can be translated in terms of logic programs. We show one example of such applications. From Immerman's theorem, "$\mathcal{NL}$ is closed under complementation" [4] (see also [10]), it follows immediately that the negation of a linear program is also linear.

In the next section, we will give several basic definitions on alternating Turing machines and the complexity classes of problems. In Section 3, we will describe a formal definition of logical query programs as a computation model. Simulations between logical query programs and logspace bounded alternating Turing machines will be presented in Section 4. In Section 5, classes of languages recognized by logical query programs will be considered.

2 Basic Concepts

2.1 Alternating Turing Machines

We assume familiarity with deterministic and nondeterministic Turing machines (DTMs and NTMs, respectively). We also adopt *alternating Turing machines* as our computation models.

ATMs are a generalization of nondeterministic Turing Machines described informally as follows. The states of an ATM are partitioned into "existential" and "universal" states. As with NTMs, we can view a computation of an ATM as a tree of configurations[1]. The *full computation tree* of an ATM M on a string w is a (possibly infinite) tree whose nodes are labeled with configurations of M on w, such that the descendants of any non-leaf node includes all of the successors of that configuration. A *computation tree* of M is a subtree of the full computation tree such that the descendants of any non-leaf node labeled by a universal configuration includes all of the successors of that configuration, and the descendants of any non-leaf node labeled by an existential configuration includes one of the successors of that configuration. An *accepting computation tree* is a finite computation tree of which all leaf nodes are accepting configurations. *M accepts w* if and only if there exists an accepting computation tree whose root node is labeled with the initial configuration of M on w. Formal definitions of ATMs are found in Chandra et al.[2].

Off-line ATMs, which have a read-only input tape and some work tapes, are defined similarly to off-line DTMs or off-line NTMs. A "random access input" variation of ATMs introduced by Ruzzo[6] is called *indexing ATMs.* We usually utilize off-line machines in our discussions. An ATM operates in *time* $T(n)$ (*tree-size* $Z(n)$) if for all acceptable input strings of length n, there is an accepting computation tree of heigl $\leq T(n)$ (respectively, size $\leq Z(n)$). An ATM operates in space $S(n)$ if for all acceptable inputs there is an accepting computation tree, each of whose nodes are labeled by a configuration using space $\leq S(n)$.

2.2 Hierarchy of Complexity Classes

We are mostly concerned with the class of problems solvable very rapidly by a parallel computer with feasible number of processors, i.e., problems which can be computed by a uniform circuit[7] with depth $O((\log n)^k)$ and polynomial size. Such a class is commonly called $\mathcal{NC}$. $\mathcal{NC}^k$ is the set of all problems solvable by a unform circuit family with depth complexity $O((\log n)^k)$ and size complexity $n^{O(1)}$. $\mathcal{NC} \stackrel{\text{def}}{=} \bigcup_k \mathcal{NC}^k$. Ruzzo showed a close relationship between uniform combinational circuits and indexing alternating Turing machines [7].

Proposition 1 (Ruzzo) *$\mathcal{NC}^k$ consists of all problems solvable by indexing ATMs in time $O((\log n)^k)$ and space $O(\log n)$, where n is the length of the input.*

Many known problems in $\mathcal{NC}$ are in some subclasses of $\mathcal{NC}^2$ *LOGCFL* is one of the most ones. *LOGCFL* consists of all sets which are logspace reducible to the class of context free languages. (Here A is *logspace reducible* to B iff there is some logspace computable function f such that for all x, $x \in A$ iff $f(x) \in B$.) Ruzzo characterized *LOGCFL* as the class of problems recognizable by tree-size bounded ATMs [6].

Proposition 2 (Ruzzo) *LOGCFL consists of all problems solvable by ATMs in space $O(\log n)$ and tree-size $n^{O(1)}$.*

$\mathcal{AC}^k$ is the class of all problems solvable by an ATM in space $O(\log n)$ and alternation depth $O((\log n)^k)$. It is known that $\mathcal{NC}^k \subseteq \mathcal{AC}^k \subseteq \mathcal{NC}^{k+1}$ for any $k = 1, 2, \ldots$. *LOGCFL* $\subseteq \mathcal{AC}^1$, and hence $\subseteq \mathcal{NC}^2$.

Hierarchy of these classes are

$$\textit{Regular Sets} \subset \mathcal{NC}^1 \subseteq \mathcal{DL} \subseteq \mathcal{NL} \subseteq LOGCFL \subseteq \mathcal{AC}^1 \subseteq \mathcal{NC}^2 \subseteq \ldots \subseteq \mathcal{NC} \subseteq \mathcal{P}$$

Here $\mathcal{DL}$ (*DLOGSPACE*) is the class solvable by a DTM in space $O(\log n)$, $\mathcal{NL}$ (*NLOGSPACE*) is the class solvable by an NTM in space $O(\log n)$, and $\mathcal{P}$ (*PTIME*) is the class solvable by a DTM in time $n^{O(1)}$ [3].

[1]The *configuration* of a Turing machine is the contents of its work tapes, the positions of its heads, and its state.

3 Logical Query Programs and the Basic Theorem Problem

We are concerned with function-free Horn logic programs. The formal definition is given as follows.

Let V be a finite set of variables, and let C be a countable set of constants. A *function-free term* is either a variable or a constant. We abbreviate it to just a "term" hereafter. Let T be the set of all terms on $V \cup C$. A *substitution* is a function $\theta : V \rightarrow T$. Applying a substitution θ to a term t, we represent the resulting term by $t\theta$. A substitution θ is called a *unifier* of two terms t_1 and t_2 if $t_1\theta = t_2\theta$. The two terms is said to be *unifiable* via θ.

An *atom* is a formula written as $p(t_1, ..., t_n)$, where p is a *predicate symbol* with arity n ($n \geq 0$), and the arguments $t_1, ..., t_n$ are terms. Let A is an atom and let $B_1, ..., B_k$ ($k \geq 0$) be zero or more atoms. A formula

$$A \leftarrow B_1, ..., B_k$$

is called a *Horn clause*. The left side of it means the conjunction of B_i's. A is called the *head* and $B_1, ..., B_k$ are called *subgoals*. A clause which has no subgoal is written just as "$A \leftarrow$ ". A *rule* is a Horn clause composed of atoms whose arguments are variables. A *basic logical query program* is a finite set of rules. We may often call it a "logic program" or a "program" for short.

A conjunction of atoms

$$A_1, ..., A_m \quad (m \geq 0)$$

is called a *goal clause*, or simply a *goal*. Variable-free goals are said to be *ground*. When $m = 1$, "A_1" is called an *unit goal*. When $m = 0$, we denote it "$\Box$" and call it an *empty goal*.

Let $\mathbf{P}_I$ be a basic logical query program. *EDB(Extensional Database) predicates* of $\mathbf{P}_I$ are predicate symbols that appear only in subgoals of rules in $\mathbf{P}_I$. Predicate symbols that appear in some rule heads are called *IDB(Intensional Database) predicates*. We use $\{p_0, p_1, ...\}$ to denote IDB predicates, $\{q_0, q_1, ...\}$ to denote EDB predicates, and $\{r_0, r_1, ...\}$ to denote predicates that may be either IDB or EDB.

An *EDB fact* of $\mathbf{P}_I$ is a clause which has no subgoal and has a head with an EDB predicate and constants as its arguments. An EDB instance is a finite set of EDB facts. A pair of a ground unit goal with an IDB predicate and an EDB instance of a program is considered as an *input* of it.

Let $\mathbf{P}_E$ be an EDB instance of a program $\mathbf{P}_I$, viz., a set of EDB facts $\{q_{j_1}(\bar{c}_1) \leftarrow, ..., q_{j_N}(\bar{c}_N) \leftarrow\}$. $\mathbf{P} \stackrel{\text{def}}{=} \mathbf{P}_I \cup \mathbf{P}_E$. (Here $\bar{c}_i$ denotes a vector of constants.) Let $N =$ "$A_1, ..., A_m$" ($m \geq 1$) be a goal, and let $C =$ "$A \leftarrow B_1, ..., B_k$" ($k \geq 0$) be a clause in $\mathbf{P}$ (either a rule in $\mathbf{P}_I$ or a fact in $\mathbf{P}_E$) such that A and A_i is unifiable via a substitution θ for some $i \in [1, m]$. Then

$$N' = (A_1, ..., A_{i-1}, B_1, ..., B_k, A_{i+1}, ..., A_m)\theta$$

is *derived* from N and C with substitution θ.

A *derivation* of N_0 from $\mathbf{P} = \mathbf{P}_I \cup \mathbf{P}_E$ is a (possibly infinite) sequence of triples $\langle N_i, C_i, \theta_i \rangle$, $i = 0, 1, ...,$ such that N_i is a goal, C_i is a clause in $\mathbf{P}$, θ_i is a substitution, and N_{i+1} is derived from N_i and C_i with substitution θ_i, for all $i > 0$. A derivation of N_0 is called a *refutation* of N_0 from P if N_ℓ is the empty goal for some $\ell \geq 0$. Such a derivation is finite and of *length* ℓ. If there exists a refutation of a goal N_0 from $\mathbf{P}$, we say that $\mathbf{P}$ *solves* N_0.

The *basic theorem problem* of a program $\mathbf{P}_I$ for an input, viz., a ground unit goal $N_0 = p_i(\bar{c}_0)$ and an EDB instance $\mathbf{P}_E$, is the question, "Does $\mathbf{P}_I \cup \mathbf{P}_E$ solves N_0?" We define the *size* of an input to be n, viz., the number of EDB facts in the input.

Let R be a refutation of N_0 from P. The *refutation tree* of R is a tree of unit goals defined as follows:

1. The root node of the tree is the initial goal N_0.
2. Leaves are empty goals.
3. In each step of derivation, $\langle N_i, C_i, \theta_i \rangle$, if a unit goal A_{ij} in N_i is unified with A_i, the head of $C_i =$ "$A_i \leftarrow B_1, ..., B_k$", via θ_i, then the node A_{ij} has directed edges to all $B_j\theta_i$'s.

The *fringe* of a refutation tree is the set of its leaves. The *fringe-size* is the number of the leaves. Note that the fringe-size of a refutation is always less than the length of it.

We say that a logical query program $\mathbf{P}_I$ is of *fringe complexity* $L(n)$ (relative to a class of EDB's $\mathcal{D}$) if any provable input (in class $\mathcal{D}$) of size n has a refutation whose fringe-size is $< L(n)$, from $\mathbf{P}_I \cup \mathbf{P}_E$.

4 Logical Query Programs and Alternating Turing Machines

4.1 An ATM Algorithm for Logical Query Programs

We will describe an algorithm for simulating a logical query program by a logspace bounded alternating Turing machine. This is based on Shapiro's naive simulation [8].

Roughly speaking, a logical query program is almost an ATM as is. Rules in the program corresponds to the next move relation of it. An initial goal and an EDB instance is regarded as an input string. The nondeterministic choice of a clause whose head unified with a goal corresponds to an existential branch of an ATM. The simultaneous satisfaction of the subgoals in the rule corresponds a universal branch.

The most important difference is that, conjunctive goals share variables, while the computations of universally forked branches of an ATM must be done independently. The key idea of Shapiro's simulation is that the final value of a shared variable is "guessed" immediately, using existential branches, before the subgoals are forked universally. Instead of the most general unifier, a ground unifying substitution, which replace all variables in the rule into constants, is chosen.

Let $\mathbf{P}_I$ be a logical query program. We first show how to encode an input, viz., an initial goal $p_i(\bar{c}_0)$ and an EDB instance $\mathbf{P}_E = \{q_{j_1}(\bar{c}_1) \leftarrow, ..., q_{j_n}(\bar{c}_n) \leftarrow\}$ into an input tape string. Since the number of IDB predicates and EDB predicates occur in $\mathbf{P}_I$ is finite and independent from the input length, we can utilize the predicate symbols as input symbols. The constants which occur in either the initial goal or the EDB facts are numbered in some appropriate order (e.g. the order of first occurrence), and represented in binary. This mapping of constants to integers is consistent because the basic theorem problem is invariant through any permutation of constants.

For example, an initial goal $p_i(\bar{c}_0)$ and an EDB instance

$$\{q_1(b,a) \leftarrow, q_2(c,c) \leftarrow, q_1(c,b) \leftarrow\}$$

is directly mapped on the input tape as

$$p_0\ (\ 0\ )\ ;\ q_1\ (\ 1\ ,\ 1\ 0\ )\ ,\ q_2\ (\ 1\ 1\ ,\ 1\ 1\ )\ ,\ q_1\ (\ 1\ 1\ ,\ 1\ )\ \#$$

Here $\boxed{0}$, $\boxed{1}$, $\boxed{(}$, $\boxed{)}$, $\boxed{,}$, $\boxed{;}$, $\boxed{p_0}$, $\boxed{q_1}$, $\boxed{q_2}$ and $\boxed{\#}$ are tape symbols. The number of distinct constants is at most n, so the indices are encoded in at most $\lceil \log_2 n \rceil$ bits. Thus the length of the input string is $O(n \log n)$.

Consider the algorithm of an ATM M below:

Algorithm 1
Given: a basic logical query program $\mathbf{P}_I$;
Input: an initial goal $p_i(\bar{c}_0)$ and an EDB instance $\mathbf{P}_E = \{q_{j_1}(\bar{c}_1) \leftarrow, ..., q_{j_N}(\bar{c}_N) \leftarrow\}$;
Output: whether $\mathbf{P}_I \cup \mathbf{P}_E$ proves $p_i(\bar{c}_0)$ or not;

procedure ASK($q(\bar{c})$: q is an EDB predicate and $\bar{c}$ is a constant vector);
  begin
    if $q(\bar{c}) \notin \mathbf{P_E}$ **then reject**
  end;
procedure DERIVE($r(\bar{c})$: a unit goal);
  begin
    if r is an EDB predicates **then**
      ASK($r(\bar{c})$);
    else
      begin
        Choose a rule "$A \leftarrow B_1, ..., B_k$" in $\mathbf{P}_I$; (existentially)
        Guess a ground substitution θ; (existentially)
        if $A\theta \neq r(\bar{c})$ **then reject**;
        for $\forall i \in \{1, ..., k\}$ **do** (universally)
          DERIVE($B_i\theta$);
      end;
  end
begin {*of the main routine*}
  Say "YES" **if** DERIVE($p_i(\bar{c}_0)$) is successfully done;
end.

The ATM M stores $\mathbf{P}_I$ in its finite control, and at its initial state it reads the initial goal $p_i(\bar{c}_0)$ written on the leftmost of the input tape. At each step of the simulation, it checks whether the current unit goal written on the work tape is composed of a EDB predicate or not. If so, it reads the input tape and checks whether the current goal is in the EDB instance. If not, it existentially chooses a rule in $\mathbf{P}_I$, and writes on its work tape a substitution θ. Next it computes $A\theta$, verifies if $A\theta = r(\bar{c})$, and erase the current goal. Then it universally chooses all subgoals B_i's in the rule and recursively dose the same for the new goal $B_i\theta$ for each i.

During th simulation, each goal is kept as a string, whose length is at most $O(\log n)$, on the work tape. The number of variables appears in a rule is at most constant, and therefore the substitution θ can be encoded in a string of length $O(\log n)$. Thus the ATM M uses space $O(\log n)$. Each call of DERIVE is done in time $O(\log n)$ and each call of ASK is done in time $O(n \log n)$. For a derivation of length L and the fringe-size F, the corresponding simulation is done with space $O(\log n)$ and tree-size $O(L \log n + F n \log n)$.

We must reminds the relation between the length and the fringe-size of a refutation [11].

Lemma 1 (Ullman et al.) *Every provable goal of a logical query program has a refutation tree whose depth is polynomial in the size of the EDB*

The length of a *tight* refutation is at most polynomial times of the fringe-size. Thus we may say $L = F \cdot n^{O(1)}$. We now have:

Theorem 1 *For every logical query program $\mathbf{P}_I$ of fringe complexity $F(n)$, there exists an off-line ATM M of space complexity $O(\log n)$ and tree-size complexity $F(n) \cdot n^{O(1)}$, such that M accepts an input (an initial goal and an EDB instance) iff the union of the EDB instance and $\mathbf{P}_I$ solves the initial goal.*

4.2 Programs simulating ATMs

Let us show the inverse of Theorem1, i.e., how a logspace bounded alternation is simulated by a basic logical query program.

One problem on the simulation is representation of the input string as an EDB instance. Since an EDB instance is an unordered set of facts, there seems no direct way to express naturally the symbol at each position on the input tape. For example, if one prepared an EDB instance

$$\{q_{j_1}(1) \leftarrow,\ q_{j_2}(2) \leftarrow,\ ..., q_{j_n}(n) \leftarrow\}$$

(here $\{1, 2, ..., n\}$ are constants representing the positions on the input tape) for an input string of length n, "$q_{j_1}q_{j_2}...q_{j_n}$", the program would not know that position 1 and position 2 is contiguous. Instead, Ullman et al. defined a class of EDB instances of the form

$$\{q_{j_1}(0,1) \leftarrow,\ q_{j_2}(1,2) \leftarrow,\ ...q_{j_n}(n-1,n) \leftarrow\}$$

and restrict EDB instances in such a class.

We adopt an EDB fact which represent, not a symbol on the input string, but a one-step transition from a configuration of the ATM to another. Such EDB facts are computed from the input string and the description (the next-move relation) of the ATM by preprocessing. A string on the work tape is embedded in a constant.

Let M be a logspace bounded alternating Turing machine. Without loss of generality, we may assume that the number of branches on every universal state is just 2. We may also assume that the symbols on auxiliary tape are "1", the blank symbol "0", "$\underline{1}$" (1 at the head position) and "$\underline{0}$" (0 at the head position).

Let S be the set of states of M. let $f_S : S \rightarrow \mathbf{N}$ be a numbering of S, where $\mathbf{N}$ is the set of nonnegative integers. We also define a numbering of $\Gamma \stackrel{\mathrm{def}}{=} \{0, 1, \underline{0}, \underline{1}\}$, $f_\Gamma : \Gamma \rightarrow \mathbf{N}$ as

$$f_\Gamma(0) = 0,\ f_\Gamma(1) = 1,\ f_\Gamma(\underline{0}) = 2,\ f_\Gamma(\underline{1}) = 3$$

Let $\mathcal{A}$ be the set of all configurations of M There exists a constant k such that, for any input string of length n, M uses at most $k(\lfloor \log_2 n \rfloor - 1)$. A configuration of M, say α, is specified by a three tuple of the current state q, the position of the input head i, and the contents (the string and the head position) of the work tape "$a_0a_1...a_{k(\lfloor \log_2 n \rfloor -1)}$", $a_j \in \Gamma$. Define a function $f_M : \mathcal{A} \rightarrow \mathbf{N}$ as

$$f_M(\alpha) \stackrel{\mathrm{def}}{=} f_S(q) + 2^{\lceil \log_2 |S| \rceil}(i + 2^{\lceil \log_2 n \rceil} \sum_{j=0}^{k(\lfloor \log_2 n \rfloor - 1)} f_\Gamma(a_j) 2^{2i})$$

We define an EDB instance $ED_M(w)$, which describes all of the one-step transition of configurations of M on w. $ED_M(w)$ is constructed as follows.

1. The initial value of $ED_M(w)$ is $\emptyset$ (the empty set).
2. If there is a universal branch from a universal configuration α to both of a configuration β and configuration γ then add an EDB fact "$q_\wedge(f_M(\alpha), f_M(\beta), f_M(\gamma)) \leftarrow$" to $ED_M(w)$.
3. If there is a branch from a existential configuration α to a configuration β then add an EDB fact "$q_\vee(f_M(\alpha), f_M(\beta)) \leftarrow$" to $ED_M(w)$.
4. For every accepting configuration δ add an EDB fact "$q_1(f_M(\delta)) \leftarrow$" to $ED_M(w)$.

Lemma 2 *For any logspace bounded ATM M, the computation of the EDB instance $ED_M(w)$ from input string w can be done by a (deterministic) logspace transducer.*

The lemma follows immediately from the fact that every possible configuration of M on w is indexed uniquely by a number of $O(\log n)$ bits. Note that the size of $ED_M(w)$ (the number of the facts) is $n^{O(1)}$.

Finally, we define the following three IDB rules as the program ID_M for M:

$$p(X) \leftarrow q_\wedge(X, Y, Z), p(Y), p(Z)$$
$$p(X) \leftarrow q_\vee(X, Y), p(X)$$
$$p(X) \leftarrow q_1(X)$$

Initial goal is set as $p(c_0)$, where c_0 is an integer which represents the initial configuration of M on w. Consider a derivation of $p(c_0)$ with $ID_M \cup ED_M(w)$. Obviously, each derivation by an EDB fact is performed for exactly one move of M, Thus, a refutation of $p(c_0)$ exists iff an acceptable computation of M on w exis . If the tree-size of the computation is $Z(n)$, the fringe-size of the refutation is $Z(n)$.

Theorem 2 *For any logspace bounded ATM M, there exists a basic logical query program ID_M , and a logspace transducer ED_M which computes an EDB of ID_M, such that ATM M accepts an input string w of length n with tree-size $Z(n)$ iff ID_M and $ED_M(w)$ solves $p(c_0)$ with fringe-size $O(Z(n))$.*

4.3 Linear Programs and Nondeterministic Turing Machines

A *linear* logical query program is one in which each rule has at most one IDB subgoal. It has been discovered independently by many people that the basic theorem problem of linear programs is in $\mathcal{NC}$ (see [11]). We show that in fact any linear program is in $\mathcal{NL}$. The simulation algorithm is almost similar to what we have shown in Section 4.1. Instead of forking subgoals universally, the machine first processes all EDB subgoals deterministicly, and then recursively derives at most one IDB subgoal.

Consider Algorithm 1. Since any rule has at most one IDB sub goal, we can rewrite "**procedure** DERIVE" as:

```
procedure DERIVE( r(c̄) : a unit goal );
  begin
        Choose a rule "A ← B, B_1, ..., B_k" in P_I; (existentially)
        {Assume that only B is an IDB subgoal.}
        Guess a ground substitution θ; (existentially)
        if Aθ ≠ r(c̄) then reject;
        for i := 1 to k do (sequentially)
           ASK( B_iθ );
        DERIVE( Bθ );
  end
```

No universal branch is required in it, and therefore, the computation can be done by a nondeterministic Turing machine.

Theorem 3 *For every linear logical query program $\mathbf{P}_I$, there exists an off-line NTM M of space complexity $O(\log$ `, such that M accepts an input (an initial goal and an EDB instance) iff the union of the EDB instance and $\mathbf{P}_I$ solves the goal.*

Similar result can also be proved for *piecewise linear* programs, in which each rule has at most one recursive subgoal.

Next we show that the inverse of Theorem 3 also holds.

Theorem 4 *For any logspace bounded NTM M, there exists an linear logical query program ID_M, and a logspace transducer ED_M which computes an EDB of ID_M, such that ATM M accepts an input string w of length n iff ID_M and $ED_M(w)$ solves $p(c_0)$.*

Proof: NTMs are regarded as ATM without universal states. The proof of Theorem 2 can be applied to NTM M. We can remove the first rule in IDB_M, since the EDB predicate $q_\wedge$ never occurs in the EDB instance ED_M. Thus the programs becomes

$$p(X) \leftarrow q_\vee(X,Y), p(X)$$

$$p(X) \leftarrow q_1(X)$$

which is linear. ■

5 Classification of Logical Query Programs

Ullman et al. proved $\mathcal{P}$-completeness of logical query programs, by Reduction from Monotone Circuit Value Problem [11]. We can show a more direct proof using the results of Section 4.

Corollary 1 (Ullman et al.) *The basic theorem problem of logical query programs is $\mathcal{P}$-complete.*

Proof: Theorem 1 and Theorem 2 imply that the basic theorem problem of logical query programs is *ALOGSPACE*-complete. From *ALOGSPACE*= $\mathcal{P}$ [2], the corollary follows. ■

Similar discussion for linear programs using Theorem 3 and Theorem 4 leads $\mathcal{NL}$-completeness of linear logical query programs.

Corollary 2 *The basic theorem problem of linear logical query programs is $\mathcal{NL}$-complete.*

A logical query program has the *polynomial fringe property* (relative to $\mathcal{D}$) if the fringe complexity of it (relative to $\mathcal{D}$) is $n^{O(1)}$. The Polynomial Fringe Theorem is stated as:

Proposition 3 (Ullman et.al [11]) *A basic logic program with the polynomial fringe property is in $\mathcal{NC}$ (precisely in $\mathcal{NC}^2$).*

We can improve it into the following strongest form from Theorem 1 and Theorem 2. A logical query program has the *superpolynomial fringe property* iff its fringe complexity is $2^{(\log n)^{O(1)}}$.

Corollary 3 *The basic theorem problem of a basic logical query program relative to a class of EDB's with the polynomial fringe property (with the superpolynomial fringe property) is LOGCFL-complete (respectively, $\mathcal{NC}$-complete).*

Proof: It follows immediately from $LOGCFL = ASPSZ(\log n, n^{O(1)})$ and $\mathcal{NC} = ASPSZ(\log n, 2^{(\log n)^{O(1)}})$, where $ASPSZ(S(n); Z(n))$ denotes languages accepted by ATMs operates in space $O(S(n))$ and tree-size $O(Z(n))$ simultaneously. ■

Classification of logical query programs is characterized via the hierarchy of subclasses of $\mathcal{P}$. We can apply many results on the theory of computational complexity, obtained on conventional models, to logical query programs. For example, we can show that the negation of any linear program is also computable by a linear program, by applying Immerman's theorem:

Proposition 4 (Immerman [4]) *$\mathcal{NL}$ is closed under complementation.*

Corollary 4 *For any linear logical query program $\mathbf{P}_I$, there exists a linear logical query program $\mathbf{P}'_I$ and a logspace transducer $f_{\mathbf{P}_I}$ such that a ground goal A is not provable from $\mathbf{P}_I$ and $\mathbf{P}_E$, iff $f_{\mathbf{P}_I}(A)$ is provable from $\mathbf{P}'_I$ and $f_{\mathbf{P}_I}(\mathbf{P}_E)$.*

Similarly, we can show that the negation of every logical query program (relative to a class of EDB's) with polynomial fringe property is computable by a logical query program relative to some class of EDB's with polynomial fringe property. This follows from the fact that $LOGCFL$ is closed under complementation [1].

6 Concluding Remarks

We have shown a new formalization of logical query languages as a theoretical model of computation, and have shown a relationship between logical query programs and tree-size bounded alternation. The fringe complexity of a program have been related to the tree-size of the corresponding ATM. Many well-known classes of problems is newly characterized via logic query programs.

We have shown that a program which have the polynomial fringe property respect to a certain EDB is $LOGCFL$-complete, but it is still open if there is a program, with the polynomial fringe property respect to all possible EDB facts, that can simulate $LOGCFL$ computation. One important difference of logical query programs from ATMs is that logical query programs cannot even count numbers if they have no default facts.

The program we have shown in Section 4.2 may be a little complicated, since it requires a precomputation by logspace transducer. More direct proof is also possible if we utilize alternating multihead finite automata instead of ATMs.

References

[1] A. Borodin, S. A. Cook, P. W. Dymond, W. L.Ruzzo band M. Tompa, "Two Applications of Complementation via Inductive Counting", *Tech. Report, IBM T. J. Watson Research Center*, Hampton, VA (1987).

[2] A. K. Chandra, D C. Kozen and L. J. Stockmeyer: "Alternation", *Journal of the Association for Computer Machinery*, **28–1**, 114–133 (Jan. 1981).

[3] S. A. Cook: A Taxonomy of Problems with Fast Parallel Algorithms", *Information and Control*, **64**, 2–22 (1985).

[4] N. Immerman: "Nondeterministic Space Is Closed under Complementation", *SIAM J. Comput.*, **17–5**, 935–938 (Oct. 1988).

[5] Y. Okabe and S. Yajima: "Parallel Computational Complexity of Logic Programs and Alternating Turing Machines", *Proc. of the International Conference on Fifth Generation Computer Science 1988*, 356–363 (1988).

[6] W. L. Ruzzo: "Tree-Size Bounded Alternation", *Journal of Computer and System Sciences*, **21**, 218–235 (1980).

[7] W. L. Ruzzo: "On Uniform Circuit Complexity" *Journal of Computer and System Sciences*, **22**, 365–383 (1981).

[8] E. Y. Shapiro: "Alternation and the Computational Complexity of Logic Programs", *J. Logic Programming*, **1**, 19–33 (1984).

[9] P. Štěpánek and O. Štěpánková: "Logic Programs and Alternation", *3rd International Conference on Logic Programming*, Lecture Notes in Computer Science, **225**, 99–106 (1986).

[10] R. Szelepcsényi: "The Method of Forcing for Nondeterministic Automata", Bulletin of EATCS, **33**, 96–100 (Oct. 1987).

[11] J. D. Ullman and A. van Gelder: "Parallel Complexity of Logical Query Programs", *27th IEEE FOCS*, 438–454 (1986).

[12] J. D. Ullman and A. van Gelder: "Parallel Complexity of Logical Query Programs", *Algorithmica*, **3**, 5–42 (1988).