Theoretical Computer Science 85 (1991) 75–95
Elsevier

An analysis of fixed-point queries on binary trees

Steven Lindell

Department of Mathematics, Haverford College, Haverford, PA 19041–1392, USA

Communicated by A. Meyer
Received June 1988
Revised June 1989

Abstract

Lindell, S., An analysis of fixed-point queries on binary trees, Theoretical Computer Science 85 (1991) 75–95.

The presence of ordering appears to play an essential role in the logical expressibility of polynomial-time queries on finite structures. By examining the expressibility and complexity of inductive queries on the class of complete unordered binary trees, we are able to show that the ability to calculate cardinality is strictly less powerful than the assumption of order.

0. Introduction

0.1. Motivation

This paper studies the expressibility of queries in fixed-point logic and the complexity of their evaluation on complete binary trees. The primary purpose of this investigation is to demonstrate that the addition of a nonlogical ordering provides an enhancement strictly beyond that of merely counting. This may enable us to understand more clearly the distinction between tractable (polynomial-time) computation and the inductive (fixed-point) queries. The reader who is unfamiliar with the idea of a query on a class of finite structures should skip ahead to Section 1 before proceeding.

On finite structures, it is well-known that all inductive queries (IND) are polynomial-time computable (PTIME) [3]:

$$\text{IND} \subseteq \text{PTIME}.$$

A proof follows directly from the semantic definition of evaluating fixed points (see

Section 1). But the converse fails miserably, and $\mathrm{IND} \neq \mathrm{PTIME}$. For example, the query

$$\|G\| \text{ is } \textit{even}$$

to determine whether or not a given graph has an odd or even number of vertices is not inductive [3]. A proof from a compactness argument can be found in [12]. A variety of other seemingly simple counterexamples share the property that they ask something about sizes. Examining their proofs reveals that known techniques (Ehrenfeucht–Fraisse games) rely on the inability of inductive queries to calculate the size of the domain (or some subdomain thereof). Examples appear in [12].

If we permit our formulas access to an underlying total ordering of the domain (e.g. by using the actual binary representation of a finite structure), then it is possible to count (see below). But in fact adding $<$ to IND allows us to express all polynomial-time queries [11, 15].

$$\mathrm{PTIME} \subseteq \mathrm{IND}(<).$$

The proof is a direct simulation of Turing machines that operate in polynomial time, with the ordering used crucially to keep track of the tape's contents, to say that one step follows another, and to read the input tape (which contains a binary encoding of the input structure).

Unfortunately, $\mathrm{PTIME} \neq \mathrm{IND}(<)$, but only because there are also formulas in $\mathrm{IND}(<)$ which express things that are not queries. For instance, consider the sentence

$$(\exists x \in V)(\exists y \in V)[x < y \wedge E(x, y)]$$

on unordered graphs with edge relation E and vertex set V. The result depends on the underlying representation of the graph (i.e. how the members of V are ordered), something which is specifically excluded from being a query. In summary,

$$\mathrm{IND} \subset \mathrm{PTIME} \subset \mathrm{IND}(<),$$

with all inclusions strict. But on those structures which contain a built-in linear order as one of their relations, inductive queries do capture all of polynomial time. Simply stated,

$$\mathrm{IND} = \mathrm{PTIME} \quad \text{for } \textit{ordered} \text{ structures.}$$

0.2. Calculating sizes on finite structures

In the presence of a built-in linear order, let us demonstrate how fixed-point queries can be used to calculate the size of any collection of elements U drawn from the domain, $0, \ldots, n-1$. First make an induction which exhausts all the elements of the domain, relying on the totality of the ordering. Then use this enumeration to count in unary the number of elements in U, using the ordering for the digit places. See, for

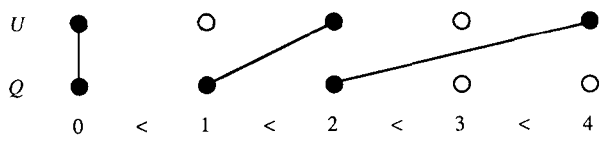

Fig. 1.

instance, Fig. 1 for $n=5$. Employ the following inductions simultaneously and let $sup(P)$ stand for one more than the largest current element in P:

$$\phi(x,P) \equiv (\forall y)[y<x \rightarrow P(y)] \qquad P \text{ does the enumerating,}$$

$$\phi(x,Q) \equiv (\forall y)[y<x \rightarrow Q(y)] \wedge U(sup(P)) \qquad Q \text{ does the counting.}$$

When the induction is complete, $sup(Q)=|U|$ ($=3$ in Fig. 1). Unfortunately, $sup(P)$ cannot be written positively in P (see Section 1 for definition), so an alternate method is required if we do not wish to appeal to [8].

Let $R(x, y)$ denote that there are more than y elements in U less than or equal to x, i.e.

$$R(x,y) \Leftrightarrow |\{z \in U \mid z \leqslant x\}| > y.$$

In particular,

$$R(n-1,i) \Leftrightarrow i<|U|.$$

Then the following formula (although confusing) defines R inductively:

$$\phi(x,y,R) \equiv (\exists z<x)[R(z,y) \vee [R(z,y-1) \wedge U(x)]],$$

where we take $R(z, y-1)$ to be vacuously true when $y=0$, even when no z exists. Precisely, $R(z,y-1) \equiv (y=0) \vee (\exists w)[wSy \wedge R(z,w)]$, where uSv if and only if v is the successor of u. At the fixed point

$$sup\{y \mid \phi^{\infty}(n-1,y)\} = |U|,$$

where *sup* is defined as one more than the maximal element. Again we require negation at this point, but the fixed-point queries are closed under negation [11].

Of course, counting is a polynomial-time query and, hence, inductive on ordered structures. The purpose of the above discussion was to provide us with concrete examples of how to count with an ordering. But is calculating sizes *all* the ordering allows us to do? Be aware that *counting* is defined as the *act* of ascertaining the total number of elements in a collection by noting one after another. This implies a calculation of size by *enumeration*. Enumeration allows one not only to determine the cardinality of a set, but also to examine each element in turn. On the other hand, methods which determine the size of a collection of elements (by no specific means) must certainly distinguish among different cardinalities but may provide no other tangible benefit.

One of the research interests behind this paper was to try to understand how an induction could be used to calculate sizes of arbitrary subsets without necessarily counting. This weaker notion can be formalized as a query with one unary relation argument, R, which always returns a result dependent only on the size of the subset under consideration. Because U is an actual parameter of the query which is not part of the structure, we write it on the right-hand side of a semicolon.

Definition 0.1. Let K be a class of structures, and R, a new unary predicate symbol not in the original signature. The query $\Phi(\bar{x}, R)$ is a *sizer* on K if for all A in K and all subsets U_1 and U_2 of A

$$|U_1| = |U_2| \Leftrightarrow \models_A (\forall \bar{a})[\Phi(\bar{a}; U_1) \leftrightarrow \Phi(\bar{a}; U_2)].$$

Or, written in a simpler fashion, $|U_1| = |U_2| \Leftrightarrow \Phi^A(U_1) = \Phi^A(U_2)$.

Interestingly enough, this definition alone is sufficient to prove the result that Φ is always an automorphism invariant set on A.

Fact. *Let Φ be a sizer on K. Then for all $A \in K$, $U \subseteq |A|$, the set $\{x \mid \Phi^A(x; U)\}$ is invariant on A.*

Proof. Let α be any automorphism of A, i.e. $\alpha[A] = A$. Then $\alpha[\Phi^A(U)] = \Phi^{\alpha[A]}(\alpha[U])$ since α is a permutation; and $\Phi^{\alpha[A]}(\alpha[U]) = \Phi^A(\alpha[U])$ since α is an automorphism. But since $|\alpha[U]| = |U|$, our assumption that Φ is a sizer leads us to the conclusion that $\Phi^A(\alpha[U]) = \Phi^A(U)$. Therefore, $\alpha[\Phi^A(U)] = \Phi^A(U)$ for all automorphisms α. □

Hence, the result of Φ^A is always one of $\|A\| + 1$ different invariant sets (there are $\|A\| + 1$ possibilities for $|U|$).

0.3. Overview

From the previous discussion, it is tempting to conjecture that the incapability to calculate sizes of arbitrary subsets is the only factor which prevents the inductive queries from encompassing all of the polynomial-time queries in the absence of an ordering.

Conjecture. *Let K be a class of finite structures. If there exists an inductive sizer on K, then every polynomial-time query on K is inductive.*

We have found a natural counterexample – the class of complete binary trees – which shows that this conjecture is *false*. In other words, there is more to the ordering than just the ability to calculate sizes.

Intuitively, complete binary trees (without distinguished left and right children) are potentially interesting because their structure falls naturally in between that of a total linear order and that of a totally unordered set. By analyzing inductive computations on the class of complete binary trees we will prove that

there is a *sizer* $\in$ IND and

there is a PTIME query $\notin$ IND

In other words, although IND$\neq$PTIME, inductions are capable of calculating cardinalities (although not by counting).

The remainder of this paper is organized into six sections:

Section 1. Background – finite structures, queries, IND and PTIME
Section 2. Coding invariant relations on complete binary trees
Section 3. First Result – finding a polytime query which is not inductive
Section 4. Adding numbers on complete binary trees
Section 5. Second Result – finding an inductive query which calculates sizes
Section 6. Conclusion – arithmetic, counting and logics for PTIME

In the background, Section 1, we discuss finite structures and queries, and present the basic definitions of polynomial-time computability and inductive definability. In Section 2, we develop a succinct representation of all automorphism invariant relations on a complete binary tree. Using this representation we analyze inductive queries on the class of complete binary trees and prove our first main result in Section 3. There we show, in Theorem 3.4, that on the class of complete binary trees there is a polytime query which is not inductive. It is interesting to note that our proof relies on diagonalization. In Section 4, we demonstrate a binary representation of numbers on the levels of a complete binary tree, and use this representation to perform addition. We prove our second main result in Section 5 and show, in Theorem 5.1, that on the class of complete binary trees there is an inductive query which calculates sizes. In the conclusion, Section 6, we note that other arithmetic operations can be performed inductively on complete binary trees. Finally, we elaborate on the connection between this research and a stronger conjecture originally posed in [11] regarding "counting" quantifiers. A negative answer to this leaves open the question of whether there is a *logic* for PTIME.

1. Background

1.1. *Finite structures*

We begin by briefly summarizing the notion of finite relational structures and queries upon them. For the necessary background in first-order logic, the reader is referred to the text [4].

Start with a finite list of predicate and constant symbols $L=\{R_1,\dots,R_k,c_1,\dots,c_l\}$ called a *signature*. A *finite structure* for the signature L,

$$A=\langle |A|, R_1^A,\dots, R_k^A, c_1^A,\dots, c_l^A\rangle,$$

consists of a finite set $|A|$ (called the *domain* of A), together with relations $R_1^A,\dots,R_k^A$ (each of a specified arity) and constants $c_1^A,\dots,c_l^A$ over $|A|$. We shall drop the superscripts for convenience whenever the context justifies it, and denote the size of A by $\|A\|$. The most common examples of finite structures are directed graphs and binary strings. A directed graph G can be represented as $\langle V, E\rangle$, where V is the set of vertices of G, and E is the binary edge relation for G. A binary string w can be represented as $\langle B, <, U\rangle$, where B is the finite set $\{1, \ldots, |w|\}$ totally ordered (in the normal order of numbers) by the binary relation $<$, and U is a unary relation which indicates the location of ones and zeros in w. That is, for $i \in B$, $U(i)$ holds if and only if the ith bit of w is one.

1.2. Queries

A *query* is a function which assigns to each structure (in a given signature L) a relation (of some fixed arity $k \geqslant 0$) on the domain of that structure in such a way as to be invariant of isomorphism. That is, $q(\bar{x})$ is called a k-ary L-query if for all L-structures A and A', and all k-tuples $\bar{a} \in |A|^k$ and $\bar{a}' \in |A'|^k$,

$$(A, \bar{a}) \cong (A', \bar{a}') \quad \Rightarrow \quad \models_A q(\bar{a}) \Leftrightarrow \models_{A'} q(\bar{a}'),$$

where the double turnstile indicates logical satisfaction. For example, this means that the result of a graph query $\{\bar{v} \mid \models_G q(\bar{v})\}$ is always independent of any labelling of the vertices of G. If $k = 0$, this means that $q \in \{\textit{false}, \textit{true}\}$ and, hence, determines a graph property (i.e. q does not change its truth value for an isomorphic copy of G). It is worthwhile noting that the subrestriction of automorphism invariance (following from isomorphism invariance) is vacuous for binary string queries, as all such structures have no nontrivial automorphisms.

1.3. Examples

Queries can be classified according to the syntactical constructs needed for their expression in some logic. Standard examples include first-order formulas, fixed-point formulas, and second-order formulas. Queries can also be classified according to the asymptotic resources required for their computation on some machine model. Standard examples here include logarithmic-space, polynomial-time and polynomial-space on a Turing machine. These resource bounds are essentially model-independent.

First-order logic is a natural language to use for expressing queries on finite structures. For example, the property of graph simplicity can be written as

$$\theta \equiv (\forall x \in V)(\forall y \in V)[E(x, y) \leftrightarrow E(y, x)] \wedge (\forall z \in V)[\neg E(z, z)].$$

θ is true if and only if the binary relation E on vertex set V is a simple graph (symmetric and loop-free). In symbols, letting $G = \langle V, E\rangle$,

$$\models_G \theta \quad \text{iff} \quad G \text{ is } \textit{simple}.$$

Since θ evaluates to either true or false, it is called a *Boolean* query. Those queries expressible explicitly by first-order formulas are classically called *elementary* queries [13]. A natural extension of first-order logic is to allow a query to be expressed recursively by a first-order formula. For example, the transitive closure of a graph whose edges are given by a binary relation P on a vertex set S can be written as

$$P^+(x, y) \equiv P(x, y) \vee (\exists z \in S)[P(x, z) \wedge P^+(z, y)].$$

P^+ is defined in terms of itself in a first-order fashion. The minimal relation satisfying this recursive definition has the property that

$$\models_G P^+(u, v) \Leftrightarrow \text{there is a (directed) path from } u \text{ to } v \text{ in } G.$$

Since the result of evaluating P^+ is a binary relation, it is called a *binary* query. Those queries expressible recursively by first-order formulas we shall call *inductive* queries (according to classical terminology [13]), and the class of such queries we denote by IND.

1.4. Polynomial-time computability

Problems which can be solved in polynomial time are said to possess a tractable algorithm. For necessary background in computational complexity, the reader is referred to [10].

Not surprisingly, the property of graph simplicity is polynomial-time computable. Also, the transitive closure of a graph is computable in polynomial time. Namely, given a binary encoding of an input graph G, there is a Turing machine which outputs a binary encoding of the transitive closure of G in polynomial time. As a matter of fact, all inductive (and, hence, elementary) queries are computable in polynomial time.

Definition. A query q is *polytime* if it can be computed by a sequential algorithm which, when given a binary encoding of the finite structure A as input, always outputs a binary encoding of the structure (A, q^A) in at most $\|A\|^c$ steps for some uniform c, where $\|A\|$ denotes the size of A.

The class of all polytime queries will be called PTIME.

1.5. Inductive definability

The fact that the transitive closure is not elementary was first shown in [6]. Because of this limitation and similar deficiencies in expressive power, augmenting first-order logic by a fixed-point operator was suggested [1]. This extension was also studied in mathematical logic as a means of formalizing the notion of an inductive definition, but the concern was primarily for fixed infinite structures [13].

This new language permits recursive first-order definitions, as in the previous example of transitive closure. The essential features shown there are:

(1) the free predicate on the left-hand side appears only positively on the right-hand side;

(2) the free variables of both sides must match.

To be formal, let $\varphi(\bar{x}, S)$ be an *S-positive* first-order formula for the signature L in which $\bar{x}$ is a tuple of free individual variables, S is a free predicate variable not in the signature L (the recursion variable) allowed only to appear positively, and $k = arity(S) = length(\bar{x})$. Over any L-structure A with additional free relation $S_0 \subseteq |A|^k$, $\varphi(\bar{x}, S)$ determines the query

$$\varphi^A(S_0) = \{\bar{a} \mid \models_A \varphi(\bar{a}; S_0)\},$$

which is monotone: $S_1 \subseteq S_2 \Rightarrow \varphi^A(S_1) \subseteq \varphi^A(S_2)$ (note that the arity of $\varphi^A(S)$ is k). Hence (dropping the superscript A for notational clarity), the *stages* of φ, $\varphi^0 = \emptyset$, $\varphi^{i+1} = \varphi(\varphi^i)$ which result from applying φ repeatedly to the empty set form a nondecreasing chain which reaches a *fixed point*

$$\emptyset = \varphi^0 \subseteq \varphi^1 \subseteq \cdots \subseteq \varphi^i \subseteq \cdots \subseteq \varphi^\infty$$

in at most $\| A \|^k$ steps since the sizes of the φ^i are increasing and bounded by the size of $|A|^k$. This limit is the least solution to satisfy the equation

$$S = \varphi(S).$$

Hence, repeated application of φ leaves φ^∞ unchanged, implying that the above chain of iterates satisfies $\varphi^n = \varphi^{n+1} = \varphi^\infty$ for all $n \geqslant \| A \|^k$. Clearly, φ^∞ does not depend on S since the fixed-point operation binds the variable S. Hence, φ^∞ is a query defined on every L-structure. The isomorphism invariance of φ^n and, hence, of φ^∞, follows from the isomorphism invariance of φ (by induction on n). This fact will be used crucially in Corollary 3.3.

The inductive queries are defined as projections (by constants) of fixed points.

Definition. Let $\varphi(\bar{x}, S)$ be an S-positive first-order formula for the signature $L \cup \{S\}$, with $S \notin L$ and $arity(S) = length(\bar{x})$. If $\bar{c}$ is a tuple of constant symbols in L and $\bar{y}$ is a tuple of variables such that $length(\bar{c}) + length(\bar{y}) = length(\bar{x}) \geqslant 0$, then the query $\varphi^\infty(\bar{c}, \bar{y})$ for the signature L obtained by instantiating some of the variables in the fixed-point query $\varphi^\infty(\bar{x})$ by constants is an *inductive* query.

It should be clear that the fixed-point semantics gives us a polynomial-time algorithm for computing inductive queries. In addition, inductive queries satisfy nice closure properties. For any signature L containing at least one constant symbol, the class of inductive queries on finite L-structures is closed under composition (nested recursion), all positive first-order operations [13], and negation [11]. These closure properties will not hold if our signatures do not have a constant symbol since nonelementary fixed points must have nonzero arity (and, therefore, a Boolean query such as *connected* would be inexpressible). The reader is referred to [13] for a thorough treatment of positive elementary induction.

2. Coding invariant relations

2.1. Preliminary definitions

Here are the definitions and notation we will be using for complete binary trees in this paper (left and right children are not distinguished).

Definition 2.1. Let $T_h = \langle V, E, r \rangle$ be the complete binary tree of height h and size $2^h - 1$ with node set V, binary parent relation E, and root r. Let $T = \{T_h \mid h \geqslant 1\}$. For instance, Fig. 2 has height $h = 4$ and size $2^4 - 1 = 15$.

The transitive closure of E yields a partial order $<$ in which $u < v \Leftrightarrow u$ is a *proper descendant of* v (v is an *proper ancestor* of u), and we write $u \parallel v$ to mean that $u \nleq v$ and $v \nleq u$ (u and v are *incomparable*). Note that $r \geqslant x$ for all x in V. In Fig. 2, $a \parallel b$, $b < c$ and $a < c$.

2.2. Depths, least common ancestors and automorphisms

Let $u \wedge v$ denote the *least common ancestor* of u and v, and define $d(v)$ as the *depth* of v (its distance from the root). Note that $d(r) = 0$. In Fig. 2, $a \wedge b = c$, $a \wedge c = c$, $b \wedge c = c$, $d(a) = 3$, $d(b) = 2$ and $d(c) = 1$.

Since T_h is a complete binary tree, it should be clear (see Fig. 2) that every one of its automorphisms can be generated by the transposition of sibling subtrees. Extending automorphisms to k-tuples of V

$$\alpha(\langle v_1, \ldots, v_k \rangle) = \langle \alpha(v_1), \ldots, \alpha(v_k) \rangle$$

induces an equivalence relation on tuples: $\bar{v} \sim \bar{v}'$ iff there is an automorphism α such that $\alpha(\bar{v}) = \bar{v}'$. For instance, in Fig. 2, $\langle a, b \rangle \sim \langle a', b' \rangle$ via the automorphism which also exchanges c and c'.

We now present two facts which will be used to succinctly describe tuples. Our first fact says that all automorphisms preserve depth and are distributive over the least common ancestor operation.

Fact. 2.2. *Let α be an automorphism. Then for all v, v_1, v_2,*

$$d(v) = d(\alpha(v)),$$

$$\alpha\text{-}(v_1 \wedge v_2) = \alpha(v_1) \wedge \alpha(v_2).$$

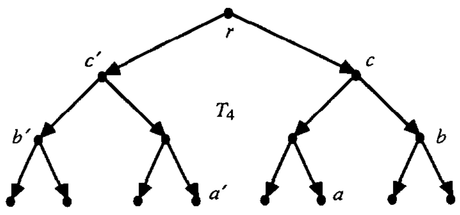

Fig. 2.

We can see clearly in Fig. 2 that $d(a)=d(a')$, $d(b)=d(b')$, $d(c)=d(c')$ and that $a' \wedge b' = c'$.

Our second fact says that any two nodes of the same depth are equivalent via some automorphism which fixes their least common ancestor.

Fact 2.3. *Suppose v and v' are of the same depth. Then there exists an automorphism α that moves v to v' and fixes every node which is not strictly below their least common ancestor. In particular, α fixes $v \wedge v'$:*

$$\alpha(v)=v', \quad d(v)=d(v'),$$

$$\alpha(u)=u, \quad \forall u \nless (v \wedge v').$$

We can see clearly in Fig. 2 that any automorphism which maps a to a' also fixes $a \wedge a' = r$ (but the root must be fixed in any case).

2.3. *Coding tuples*

It follows that any *node* v can be described uniquely (up to automorphism) by its *depth*

$$v \sim v' \Leftrightarrow d(v)=d(v'),$$

i.e. the depth of v is a *complete automorphism invariant* for v.

Similarly, the *triple of integers*

$$\langle d(v_1), d(v_1 \wedge v_2), d(v_2)\rangle$$

defines a complete automorphism invariant for any *pair of nodes* $\langle v_1, v_2\rangle$. It is slightly more complicated because automorphisms also preserve the depths of least common ancestors. Refer to the three examples $\langle c,c\rangle \sim \langle c',c'\rangle$, $\langle b,c\rangle \sim \langle b',c'\rangle$ and $\langle a,b\rangle \sim \langle a',b'\rangle$ in Fig. 2.

Extending this idea to k-tuples requires the depths of all the least common ancestor pairs.

Definition 2.4. Let

$$\begin{aligned}\langle v_1,\ldots,v_k\rangle^* &= \langle d(v_i \wedge v_j)\rangle_{i \leqslant j}\\ &= \langle d(v_1 \wedge v_1),\ldots,d(v_1 \wedge v_k),\ldots,d(v_i \wedge v_i),\ldots,d(v_i \wedge v_k),\ldots,d(v_k \wedge v_k)\rangle.\end{aligned}$$

Denote this invariant by $\bar{v}^*$ and its length by $k^* = k(k+1)/2$. The length is important because it depends only on k and not on the size or depth of the tree (we saw previously that $1^*=1$ and $2^*=3$). The following lemma proves that $\bar{v}^*$ is a complete automorphism invariant for $\bar{v}$.

Lemma 2.5. *For all k-tuples $\bar{v}$ and $\bar{v}'$, $\bar{v}^* = \bar{v}'^* \Leftrightarrow \bar{v} \sim \bar{v}'$.*

Proof. ($\Leftarrow$) Obvious from Fact 2.2. ($\Rightarrow$) By induction on k. The basis $k=1$ follows immediately from Fact 2.3. For the induction step, consider $\bar{v}^*=\bar{v}'^*$ for $k>1$. Let $u=v_1 \wedge \ldots \wedge v_k$ and $u'=v'_1 \wedge \ldots \wedge v'_k$. Observe that

$$d(u)=d(v_1 \wedge \ldots \wedge v_k)=min_{i,j}\, d(v_i \wedge v_j),$$

and

$$d(u')=d(v'_1 \wedge \ldots \wedge v'_k)=min_{i,j}\, d(v'_i \wedge v'_j).$$

Since $d(v_i \wedge v_j)=d(v'_i \wedge v'_j)$ by hypothesis, this forces $d(u)=d(u')$. So, by Fact 2.3, we can find an automorphism mapping u to u' which preserves all the depth numbers in $\bar{v}^*$. Hence, we might as well assume that $u=u'$ from the start. Now, there are two cases.

Case I: If some $v_i=u$, then $d(v_i)=d(u)=d(v'_i)$ implies $v'_i=u$ also. By Definition 2.4,

$$\langle v_1,\ldots,v_{i-1}, v_{i+1},\ldots,v_k\rangle^*=\langle v'_1,\ldots,v'_{i-1},v'_{i+1},\ldots,v'_k\rangle^*$$

since the equality constraints in the hypothesis ($\bar{v}^*=\bar{v}'^*$) are a superset of those required. So by induction hypothesis there is an automorphism β which witnesses that

$$\langle v_1,\ldots,v_{i-1}, v_{i+1},\ldots,v_k\rangle \sim \langle v'_1,\ldots,v'_{i-1}, v'_{i+1},\ldots,v'_k\rangle.$$

But since all these nodes are descendants of u, it is easy to see that β must fix $u=v_i$ (i.e. $\beta(v_i)=v'_i$). Hence, $\beta(\bar{v})=\bar{v}'$, and β also witnesses $\bar{v}'\sim\bar{v}$.

Case II: If for all i, $v_i<u$, then let u_1 and u_2 be the two children of u. We can partition the set of indices $I=\{1,\ldots,k\}$ into two nontrivial disjoint subsets $I_1=\{i\,|\,v_i\leqslant u_1\}$ and $I_2=\{i\,|\,v_i\leqslant u_2\}$ such that $I_1\cup I_2=I$ and $I_1\cap I_2=\emptyset$; and similarly, with I'_1 and I'_2 defined from the v'_i. Observe that if we are given any pair of nodes v_i and v_j, $d(v_i \wedge v_j)=d(u)$ iff i and j lie in distinct partitions of I. Similarly, for any pair of nodes v'_i and v'_j. Since $d(v_i \wedge v_j)=d(v'_i \wedge v'_j)$ by hypothesis, this implies that $I_1=I'_1$ (it may be necessary to automorphically transpose u_1 and u_2 to ensure that v'_1 and v_1 lie in the same subtree). Now use induction to find an automorphism γ_1 such that $\gamma_1(v_i)=v'_i$ for each $i\in I_1$ and γ_1 fixes the subtree of u_2 (by Fact 2.3). Similarly, find an automorphism γ_2 such that $\gamma_2(v_i)=v'_i$ for each $i\in I_2$ and γ_2 fixes the subtree of u_1. So the composition $\gamma_1\circ\gamma_2(\bar{v})=\bar{v}'$ witnesses $\bar{v}\sim\bar{v}'$. $\square$

2.4. Compacting relations

Automorphisms can be extended naturally to act on entire sets $\alpha[R]=\{\alpha(\bar{x})\,|\,\bar{x}\in R\}$.

Definition 2.6. A relation $R_0\subseteq|T_h|^k$ is *invariant* on T_h if it is fixed by every automorphism, i.e. for all automorphisms α of T_h,

$$\alpha[R_0]=R_0.$$

For instance, the binary parent relation E of T_h is invariant by definition.

Since invariant relations are simply composed of equivalence classes of tuples, we can succinctly describe them using the complete automorphism invariants for tuples explained previously. We simply extend the star operator to relations.

Definition 2.7. Let $R_0^* = \{\bar{v}^* \mid \bar{v} \in R_0\}$.

This provides a unique encoding of invariant relations (R_0^* simply tells which equivalence classes belong to R_0). The next fact follows immediately from Lemma 2.5.

Fact 2.8. *If R_1 and R_2 are invariant relations, then*

$$R_1 = R_2 \Leftrightarrow R_1^* = R_2^*.$$

3. Separating IND from PTIME on complete binary trees

In this section, we show that the class of inductive queries is strictly contained within the class of polytime queries on the class of complete unordered binary trees. Consider an auxiliary class of structures.

Definition 3.1. Let $H = \{H_h \mid h \geqslant 1\}$, where

$$H_h = \langle \{0, \ldots, h-1\}, \mathit{Succ}, \mathit{zero} \rangle,$$

Succ is a binary successor relation satisfying $\mathit{Succ}(i, j) \Leftrightarrow i+1=j$, $0 \leqslant i, j \leqslant h-1$, and *zero* is a constant interpreted as being the unique element with no predecessor (i.e. 0). Note that every relation on H_h is invariant since H_h has no nontrivial automorphisms.

Combined with the compact encoding of invariant relations given by Definition 2.7, we see that if R_0 is a k-ary relation on T_h, then R_0^* will be a k^*-ary relation on H_h (since $d(v) \in \{0, \ldots, h-1\}$ for all $v \in T_h$). Hence, every invariant relation on T_h is in one-to-one correspondence with a wider relation on H_h whose width does not depend on h. Since this is an exponentially smaller structure, we can "store" invariant relations on binary trees very compactly.

Because each stage of an induction is an invariant relation, it will be possible to rewrite inductions on complete binary trees T_h and simulate them on exponentially smaller structures H_h. In other words, given a first-order formula $\varphi(\bar{x}, S)$, we will syntactically construct a first-order formula $\varphi^*(\bar{\imath}, S^*)$ such that, uniformly for all h, the computation

$$S^* \leftarrow \emptyset, \quad S^* \leftarrow \{\bar{\imath} \mid H_h \models \varphi^*(\bar{\imath}, S^*)\}$$

will preserve the semantics of the computation

$$S \leftarrow \emptyset, \quad S \leftarrow \{\bar{x} \mid T_h \models \varphi(\bar{x}, S)\}.$$

Table 1

Original	Translation	Reason
$root^*$	$zero$	$d(root)=0$
$[x=y]^*$	$i=j=k$	$d(x)=d(x \wedge y)=d(y)$
$[E(x, y)]^*$	$Succ(i)=Succ(j)=k$	$d(x)+1=d(x \wedge y)+1=d(y)$
$[S(\bar{x})]^*$	$S^*(i_1, \ldots, i_{m^*})$	$arity^*(S)=arity(S^*)$
$[\neg\phi]^*$	$\neg\phi^*$	$\bar{R}^*=\overline{R^*}$
$[\psi_1 \wedge \psi_2]^*$	$\psi_1^* \wedge \psi_2^*$	$[R_1 \cap R_2]^*=R_1^* \cap R_2^*$
$[(\exists x)\theta(x, \bar{y})]^*$	$(\exists i_0, \ldots, i_n)\theta^*(\bar{\imath}, \bar{\jmath})$	$(x, \bar{y})^*=\langle d(x)\rangle \hat{}\ \langle d(x \wedge y_l)\rangle_{l=1,\ldots,n} \hat{}\ \bar{y}^*$

Lemma 3.2. *Let S be a predicate symbol of arity m, and let S^* be a predicate symbol of arity m^*. If $\varphi(\bar{x}, S)$ is an S-positive first-order formula for the signature $\{E, root, S\}$, then there exists an S^*-positive first-order formula $\varphi^*(\bar{\imath}, S^*)$ for the signature $\{Succ, zero, S^*\}$ such that for all invariant relations S_0 on T_h*

$$\{\bar{x}_0 \mid T_h \models \varphi(\bar{x}_0, S_0)\}^* = \{\bar{\imath}_0 \mid H_h \models \varphi^*(\bar{\imath}_0, S_0^*)\}$$

Proof. Construct φ^* inductively from the syntax of φ by Table 1, and think of the variables i, j, k as representing $d(x)$, $d(x \wedge y)$, $d(y)$, respectively, where x, y, z are variables interpreted as ranging over $|T_h|$, i, j, k are variables interpreted as ranging over $|H_h| = \{0, \ldots, h-1\}$, and $\bar{y}$ is an n-tuple of variables.

It should be clear from the syntax that φ^* is always an S-positive first-order formula. It remains to be shown that φ^* preserves the semantics of φ, i.e. for all invariant relations S_0

$$[\varphi^{T_h}(S_0)]^* = (\varphi^*)^{H_h}(S_0^*)$$

(this is just a shorthanded rewriting of the conclusion). We accomplish this by applying the reasons shown in Table 1.

The basis case for the constant is obvious, as for all h

$$[(root)^{T_h}]^* = d(root) = 0 = (zero)^{H_h}.$$

The basis case for the equality relation follows because if x is equal to y, then $x = x \wedge y = y$ are all of the same depth. Hence,

$$\begin{aligned}\{\langle x, y\rangle^* \mid T_h \models (x=y)\} &= \{\langle d(x), d(x \wedge y), d(y)\rangle \mid x \text{ equals } y\}\\ &= \{\langle i, i, i\rangle \mid 0 \leqslant i \leqslant h-1\}\\ &= \{\langle i, j, k\rangle \mid H_h \models (i=j=k)\}.\end{aligned}$$

The basis case for the edge relation follows because if x is the parent of y, then the depths of x and $x \wedge y$ are one less than the depth of y. Hence,

$$\begin{aligned}\{\langle x, y\rangle^* \mid T_h \models E(x, y)\} &= \{\langle d(x), d(x \wedge y), d(y)\rangle \mid x \text{ is the parent of } y\}\\ &= \{\langle i, i, i+1\rangle \mid 0 \leqslant i < h-1\}\\ &= \{\langle i, j, k\rangle \mid H_h \models Succ(i) = Succ(j) = k\}.\end{aligned}$$

For the basis case of the recursion variable, it is only necessary to check that the arity of S^* is correct, i.e.

$$S(x_1, \ldots, x_m) \text{ is } m\text{-ary} \quad \text{and} \quad S^*(i_1, \ldots, i_{m^*}) \text{ is } m^*\text{-ary}.$$

The inductive cases of negation and conjunction follow trivially from the reasons given. The inductive case of quantification will follow since the expansion of $\langle x, \bar{y}\rangle^*$ in accordance with Definition 2.4 is a frontal extension of $\bar{y}^*$ by $n+1$ additional members ($n = length(\bar{y})$). To prove that

$$\{\bar{y}^* \mid T_h \models \exists x\theta(x, \bar{y})\} = \{\bar{j} \mid H_h \models \exists i_0, \ldots, i_n \theta^*(\bar{\imath}, \bar{j})\},$$

we need to show that quantifying over x is equivalent to quantifying over the i's. Our induction hypothesis is $\{\langle x, \bar{y}\rangle^* \mid T_h \models \theta(x, \bar{y})\} = \{\langle i_0, \ldots, i_n, \bar{j}\rangle \mid H_h \models \theta^*(\bar{\imath}, \bar{j})\}$. So, clearly, $T_h \models \exists x\theta(x, \bar{y}) \Rightarrow H_h \models \exists \bar{\imath}\theta^*(\bar{\imath}, \bar{j})$ by setting

$$i_0 = d(x),$$

$$i_l = d(x \wedge y_l) \text{ for } l = 1, \ldots, n,$$

$$\bar{j} = \bar{y}^*.$$

Conversely, $H_h \models \exists \bar{\imath}\theta^*(\bar{\imath}, \bar{j}) \Rightarrow T_h \models \exists x\theta(x, \bar{y})$ since if $H_h \models \theta^*(\bar{\imath}, \bar{j})$ holds, then by induction hypothesis $\langle \bar{\imath}, \bar{j}\rangle = \langle x, \bar{y}\rangle^*$ for some x, $\bar{y}$ satisfying $T_h \models \theta(x, \bar{y})$, which implies that $T_h \models \exists x\theta(x, \bar{y})$. □

Corollary 3.3. *For all h,*

$$T_h \models \varphi^\infty(root, \ldots, root) \Leftrightarrow H_h \models (\varphi^*)^\infty(zero, \ldots, zero).$$

Proof. By mathematical induction on the number of stages, it is easy to see that

$$[(\varphi^\infty)^{T_h}]^* = [(\varphi^*)^\infty]^{H_h}.$$

The details go through precisely since $\emptyset^* = \emptyset$ (basis) and because every iterate φ^i is an invariant relation (induction step). To complete the proof, just notice that $\langle root, \ldots, root\rangle^* = \langle zero, \ldots, zero\rangle$. □

Since $(\varphi^*)^\infty$ can be computed in time h^c ($h = \| H_h \|$), φ^∞ can be computed in time $\log^c n$ ($n = \| T_h \| = 2^h - 1$). Hence, by evaluating the corresponding "star" induction on H, every Boolean inductive query on T is in fact "polylogarithmic"-time computable (using a random-access Turing machine). It comes then as no surprise that there are noninductive polynomial-time queries on T.

Theorem 3.4. *There is a polytime query on T which is not inductive.*

Proof (*By diagonalization against the polylogtime queries*). Let L_0 be a binary language (a set of finite strings over the alphabet $\{0, 1\}^*$) which is in DTIME(2^{2^n}) and not in DTIME(2^{cn}) for any c [9]. The existence of L_0 can also be derived from Theorem 12.9 of [10]. Define q_0 to be the Boolean query

$$T_h \models q_0 \Leftrightarrow \text{the binary expansion of } h \text{ is a string in } \{1\}\,.\,L_0,$$

where leading zeros are omitted.

First, we claim that q_0 is polytime. Construct a Turing machine M which when it is given (the encoding of) T_h as input ($\|T_h\| = n$) computes its height $h = \log(n+1)$ and writes it in binary on a separate work tape suppressing all leading zeros together with the most significant digit one. Now, M simulates the doubly exponential time machine for L_0 and accepts if and only if this computation accepts. Since the binary encoding of h is of length about $\log h$, the whole algorithm runs in linear time

$$O(2^{2^{\log\log n}}) = O(n).$$

Second, we claim that q_0 is not inductive. Towards a contradiction, suppose q_0 is inductive. Then consider the decision algorithm for L_0 which, when given a binary string w of length m, prefixes a one to it and generates the structure H_h, where h is $1\,.\,w$ written in binary ($2^m \leqslant h < 2^{m+1}$). Now compute the result of q_0^* on H_h, and answer yes if and only if it evaluates to true. By Corollary 3.3, this happens exactly when $T_h \models q_0$. But since $\|H_h\| = h$, the time required for the entire algorithm is $O(h^c) = O(2^{cm})$: a contradiction. □

4. Adding numbers on complete binary trees

4.1. Representation of numbers

With some further definitions, we shall make a precise representation of numbers by means of invariant subsets on complete unordered binary trees.

If m is a number between 0 and $2^h - 1$, let the binary expansion of m be denoted simply as

$$m = b_{h-1} \ldots b_0, \quad b_i \in \{0, 1\},$$

where leading zeros are included. For $0 \leqslant i < h$, let the levels of the tree T_h be denoted by $L_i = \{v \in V \mid d(v) = i\}$, and note that $|L_i| = 2^i$.

Definition 4.1. For all $m = b_{h-1} \ldots b_0$, let

$$\rho(m) = \bigcup_i \{L_i \mid b_i = 1\} \textit{ represent } m \text{ on } T_h.$$

The invariant subset $\rho(m)$ has the nice property

$$|\rho(m)| = \sum_i (b_i \cdot 2^i) = m.$$

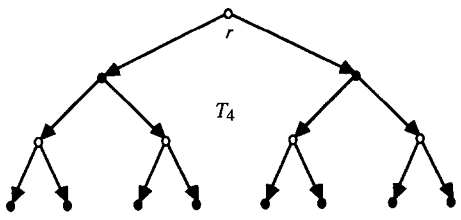

Fig. 3.

In particular, $\rho(0)=\emptyset$, $\rho(1)=\{root\}$ and $\rho(2^h-1)=V$. In Fig. 3, the open circles illustrate an example for $\rho(5)$.

Our first task is to show that addition can be defined inductively using this binary representation. Before proceeding, it will be instructive to consider again the class of structures H as defined in Definition 3.1. By identifying each i in H_h with L_i in T_h and vice versa, we can think of numbers between 0 and 2^{h-1} in binary on H_h by adopting the convention that any binary number $m=b_{h-1}\dots b_0$ can be used to talk about the unary relation on H_h:

$$m(i)=true \Leftrightarrow b_i=1, \quad i\in\{0, 1,\dots, h-1\}.$$

Similarly, identify 0 and 1 with the Boolean values *false* and *true*, respectively.

4.2. Addition

With the preceding notation, we can express the ordinary schoolbook addition algorithm which adds m to n in binary with a carry-in o. The carry bits are written recursively on H as

$$\begin{aligned} carry(i) \equiv\ & [m(i) \wedge n(i)] \vee \\ & [m(i) \vee n(i)] \wedge [carry(i-1) \vee [(i=zero) \wedge o]], \end{aligned}$$

where o takes the place of $carry(-1)$, the incoming carry for the least significant bit. Once the carry bits are known, it is a simple matter to calculate the bitwise sum:

$$sum(i) \equiv m(i) \oplus n(i) \oplus [carry(i-1) \vee [(i=zero) \wedge o]],$$

where $\oplus$ is the symbol for the exclusive-or. These equations reflect addition by the familiar binary full-adder with a rippled carry. Hence, the formulas above define a query on H satisfying

$$H_h \vDash sum(i;\ m, n, o) \Leftrightarrow (m+n+o)(i).$$

With exactly the same idea we can express addition inductively on $T=\{T_h \mid h\geqslant 1\}$. This means that given the representations, $\rho(m)$, $\rho(n)$ and o, of two numbers m, n and a carry-in o, we will calculate the representation $\rho(m+n+o)$ of their sum $(m+n+o)$.

Lemma 4.2. *The unary query* $sum(x;\ M,\ N,\ O)$ *defined on* T_h *as satisfying*

$$\{x \mid T_h \models sum(x;\ \rho(m),\ \rho(n),\ o)\} = \rho(m+n+o),$$

$$o \in \{false,\ true\}, \quad m,\ n \in \{0, \ldots, 2^h - 1\}$$

is inductive (remember that 0 *is false and* 1 *is true).*

Proof. Let M and N be unary predicates for $\rho(m)$ and $\rho(n)$ respectively, and let O be a nullary predicate for the carry-in (O is interpreted as *true* iff there is a carry-in). Using the standard schoolbook method for addition, the following formulas define the bitwise carry and sum:

$$carry(x) \equiv [M(x) \wedge N(x)] \vee$$

$$[M(x) \vee N(x)] \wedge [[(\exists y)E(y,x) \wedge carry(y)] \vee [(x = root) \wedge O]],$$

$$sum(x) \equiv M(x) \oplus N(x) \oplus [[(\exists y)E(y,x) \wedge carry^{\infty}(y)] \vee [(x = root) \wedge O]].$$

Since the inductive queries are closed under composition, *sum* is inductive. □

5. Calculating sizes inductively on complete binary trees

In this section, we finish our results and show that there is an inductive way to calculate the size of any subset of a complete binary tree, even though we cannot count in an enumerative fashion.

The idea is to produce a unary query which, when given any subset U of a complete binary tree, calculates $\rho(|U|)$, the invariant set representing the cardinality of U. This will then satisfy the requirements of Definition 0.1. For instance, for any subset consisting of exactly five elements, the result would look like Fig. 3. Recall that we write U, the actual parameter for the arbitrary subset, on the right-hand side of a semicolon.

Theorem 5.1. *Let* R *be a unary predicate symbol. Then the query* $\Phi(x, R)$ *for the signature* $\{E,\ root,\ R\}$ *defined on every complete binary tree* T_h *by*

$$\{v \mid T_h \models \Phi(v;\ U)\} = \rho(|U|), \quad \forall U \subseteq |T_h|$$

is an inductive sizer.

Proof. The set equation which defines Φ clearly satisfies the constraints of Definition 0.1 for being a sizer since $\rho(|U|)$ is unique for each choice of $|U|$. To show that Φ is inductive, the idea is to use a binary relation $Q(y, z)$ which tags each node y with the unary relation $Q_y = \{z \mid Q(y, z)\}$. If we intend Q_y to be the invariant subset representing the number of nodes below y which satisfy U,

$$Q_y = \rho(|\{x \leqslant y:\ x \in U\}|),$$

then the idea is to calculate $Q(y, z)$ recursively up the tree from the leaves to the root. We do this by tagging each parent with the sum of the tags of its children (if any) together with a 1 or 0 depending on whether or not U holds at the parent. We cannot choose the order of summation for the two children, but this does not matter since addition is commutative. After h iterations, Q_{root} will represent the total number of nodes in the tree satisfying U. At each stage, the induction passes along information at each node as to the size of the subset seen so far (below it) to its parent. Define the formula Φ with recursion variable Q and formal unary parameter R:

$\Phi(y, z, Q; R) \equiv$

Formula	*Comments*
$(\exists u)(\exists v)[u \neq v \wedge E(y, u) \wedge E(y, v)$	If u and v are the children of y,
$\wedge\ sum(z;\ Q_u,\ Q_v,\ R(y))]$	then $\#y := \#u + \#v + R(y)$.
$\vee\ (\forall u)[\neg E(y, u)$	If y is a leaf,
$\wedge\ z = root \wedge R(y)]$	then $\#y := R(y)$.

Here $\#x$ is an abbreviation for $|Q_x|$ and $R(y)$ is identified with 1 if $R(y)$ holds, and with 0 otherwise. The last line of the formula tags each leaf y with a 1 if and only if y is in R; otherwise, it is tagged with a 0. The first line says that u and v are the two children of an internal node y. The middle line tags y with the tag of u plus the tag of v, plus an additional 1 if y is in R.

It is simple to verify by induction on the levels of the tree that, for all subsets U, and for all y, $\{z \mid \Phi^{\infty}(y, z; U)\}$ is an invariant subset satisfying

$$|\{z \mid \Phi^{\infty}(y, z; U)\}| = |\{w \leqslant y \mid w \in U\}|.$$

Therefore,

$$\{z \mid \Phi^{\infty}(root, z; U)\} = \rho(|U|)$$

is the desired query. But we are not done because, strictly speaking, $\Phi(y, z, Q; U)$ is not a Q-positive formula since Q appears negatively in $sum(z; Q_u, Q_v, U(y))$ for the exclusive-or operation. To solve this problem, we monotonize the induction by adding a unary predicate $done(y)$ and delay evaluation of $\#y$ until all the children of y satisfy $done$. This means that Q_y will remain empty until the stage at which it attains its final value, ensuring that the induction determined by Φ is monotone. We can then appeal to the result in [8], where it is shown that on finite structures the fixed point of any monotone induction is in fact inductive. □

6. Conclusion

Once we have shown that sizes of unary relations can be calculated inductively on complete binary trees, similar proof techniques allow us to calculate the cardinality of binary relations and, indeed, relations of any arity. The idea (for a binary relation R) is to calculate the size of the unary relation $\{y \mid R(x, y)\}$ for each node x

$$n_x = |\{y \mid R(x, y)\}|$$

using Theorem 5.1. Then just recursively sum the n_x's over all nodes x in the binary tree in the same manner. It gets messy because the size may be large enough to require two digits, necessitating a fancier two-digit *sum* query, and we leave the details to the reader.

And once we have shown that addition is inductive on the class of complete binary trees, it is not difficult to show that other arithmetic operations such as multiplication are also inductive since they can be defined recursively from addition. Hence,

size calculation and arithmetic operations are inductive

on the class of complete binary trees. Yet by Theorem 3.4

$$\text{IND} \neq \text{PTIME}$$

on the class of complete binary trees. Therefore, we can conclude that

inductive size and arithmetic calculation $\not\Rightarrow$ IND = PTIME.

This means that the calculation of cardinality does not require the full power that an ordering has to offer. But there is still a broader question which has not been answered. The addition of ordering to fixed-point logic is unsatisfactory, primarily, because it introduces formulas which do not determine queries (isomorphism invariant) on the original unordered structure. But is there any isomorphism invariant operation which can be added to fixed-point logic so that the resulting logic captures all of the polytime queries? Loosely speaking, the question is:

Is there a logic for PTIME?

This question is equivalent to knowing if PTIME is recursively enumerable, and has also been raised in [3] and [7].

One possibility is to include counting into fixed-point logic as in [11], where it is proposed to form a two-sorted structure from the original structure A by adding an additional domain of *numbers*

$$\{0, 1, 2, \ldots, \|A\| - 1\}$$

of the same size with the usual successor relation. The only connection between the two domains are *counting* quantifiers which are used like

$$(\exists i \ x\text{'s})\varphi(x, \bar{y}),$$

where i ranges over the number domain. On the surface, this appears to only allow one to calculate cardinalities, e.g.

$$(\exists i)\,[(\exists i\ x\text{'s})\,\varphi(x) \wedge (\exists i\ x\text{'s})\,\psi(x)]$$

says that the cardinalities of the subsets determined by φ and ψ are equal. But since formulas can contain both regular variables $x, y, z, \ldots$ and number variables $i, j, k, \ldots$ it is possible to create inductions which use both simultaneously:

$$\varphi(i, x, S) \equiv (\forall y) S(i-1, y) \vee (\exists i\ z\text{'s})(z \neq z).$$

In particular, this is an example of an induction whose length is always $\|A\|$, even with no relations on the structure. Let us denote this class of queries by IND + *counting*. In fact, on the class of complete binary trees, it is easy to show that

$$\text{IND} + \textit{counting} = \text{PTIME}.$$

The question of whether or not the above identity holds on the class of all (unordered) structures was recently answered in [2], showing that fixed-point logic augmented by full counting is not all of polynomial time.

Acknowledgment

This work comes from the author's dissertation. He thanks both of his advisors, Sheila Greibach and Yiannis Moschovakis, for their time and patience. Additional appreciation is expressed to Neil Immerman for the useful discussions which helped clarify some of the consequences of these results, and to the referees for indicating the need for more concise proofs and motivational material.

References

[1] A.V. Aho and J.D. Ullman, Universality of data retrieval languages, in: *Proc. 6th Symp. on the Principles of Programming Languages* (1979) 110–117.
[2] J. Cai and N. Immerman, An optimal lower bound on the number of variables for graph identification, in: *Proc. 30th Ann. Symp. on the Foundations of Computer Science* (1989) 612–617.
[3] A. Chandra and D. Harel, Structure and complexity of relational queries, *J. Comput. System. Sci.* **25**(1) (1982) 99–128.
[4] H.B. Enderton, *A Mathematical Introduction to Logic* (Academic Press, New York, 1972).
[5] R. Fagin, Generalized first-order spectra and polynomial time recognizable sets, in: R. Karp, ed., *Complexity of Computation, Proc. SIAM-AMS #7* (1974) 43–73.
[6] R. Fraisse, *Course in Mathematical Logic, Vol. 1: Relation and Logical Formula* (D. Louvish, translation), 1973.
[7] Y. Gurevich, Logic and the challenge of computer science, in: E. Borger, ed., *Current Trends in Theoretical Computer Science* (Computer Science Press, Rockville, MD).
[8] Y. Gurevich and S. Shelah, Fixed-point extensions of first-order logic, *Ann. Pure Appl. Logic* **32** (1986) 265–280.

[9] J. Hartmanis and R.E. Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965) 285–306.

[10] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).

[11] N. Immerman, Relational queries computable in polynomial time, *Inform. and Control*, **68** (1986) 86–104.

[12] S. Lindell, The logical complexity of queries on unordered graphs, Ph.D. Dissertation, University of California, Los Angeles, 1987.

[13] Y.N. Moschovakis, *Elementary Induction on Abstract Structures* (North-Holland, Amsterdam, 1974).

[14] M. De Rougemont, Second-order and inductive definability on finite structures, Ph.D. Dissertation, University of California, Los Angeles, 1983.

[15] M.Y. Vardi, Complexity of relational query languages, in: *Proc. 14th ACM Symp. on the Theory of Computing* (1982) 137–146.