Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
All Rights Reserved by Academic Press, New York and London
Vol. 25, No. 1, August 1982
Printed in Belgium

Structure and Complexity of Relational Queries

ASHOK CHANDRA AND DAVID HAREL

Computer Sciences Department, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598

Received May 4, 1982

This paper is an attempt at laying the foundations for the classification of queries on relational data bases according to their structure and their computational complexity. Using the operations of composition and fixpoints, a $\Sigma - \Pi$ hierarchy of height ω^2, called the fixpoint query hierarchy, is defined, and its properties investigated. The hierarchy includes most of the queries considered in the literature including those of Codd and Aho and Ullman. The hierarchy to level ω characterizes the first-order queries, and the levels up to ω are shown to be strict. Sets of queries larger than the fixpoint query hierarchy are obtained by considered the queries computable in polynomial time, queries computable in polynomial space, etc. It is shown that classes of queries defined from such complexity classes behave (with respect to containment) in a manner very similar to the corresponding complexity classes. Also, the set of second-order queries turns out to be the same as the set of queries defined from the polynomial-time hierarchy. Finally, these classes of queries are used to analyse a set of queries defined from language considerations: those expressible in a programming language with only typed (or ranked) relation variables.

1. INTRODUCTION

In recent years the theory of relational data bases has received a great deal of attention in computer science research. One of the central topics of research in this area is that of queries (i.e., functions from data bases to relations) and query languages (i.e., languages for expressing such functions).

Since Codd's early work on relational data bases [3], many diverse query languages and associated classes of queries have been suggested. One encounters the first-order relational calculus and the relational algebra of Codd [4] (see also [15]), the conjunctive queries of Chandra and Merlin [9] and the tableau queries of Aho *et al.* [1]. Zloof [25] has suggested augmenting the first-order queries with a transitive closure operator, and Aho and Ullman [2] have augmented the relational algebra with a least fixpoint operator. Some fixpoint queries are also obtained by querying in Kowalski's language of logic programs [15, 17, 22]. Chandra and Harel [6] defined the general class CQ of all computable queries which, in some sense, is the largest interesting such set. In [6] a query language is defined and is shown to express

0022-0000/82/040099-30$02.00/0
Copyright © 1982 by Academic Press, Inc.
All rights of reproduction in any form reserved.

precisely the queries in CQ. While this query language indeed subsumes all the aforementioned languages, and hence CQ contains all the queries expressible in such languages, [6] does not provde a global framework within which these and other classes of queries can be investigated on common ground. The purpose of this paper is to lay the foundations for such a framework making possible the classification and comparison of query languages and their associated classes of queries according to their structure or the complexity of their computations. Definitions of fundamental concepts are proposed, accompanied by various results and open problems for future work.

The paper consists of two parts: algebraic and complexity-theoretic in nature, respectively. First, the basic set E of existential queries is defined. This set subsumes the conjunctive [9] and tableau queries [1]. From E, larger sets of queries can be obtained by using three operations on queries: complementation, composition, and least fixpoint. These operations, when applied to E, yield a natural hierarchy of sets of queries which we call the *fixpoint query hierarchy*, or the *fixpoint hierarchy* for short. This is a $\Sigma - \Pi$ hierarchy of height ω^2, in which the first ω steps (which we call the *first-order query hierarchy*) constitute a structural clasification of the first-order queries (and hence also of the relational algebra queries) of Codd [4]. We show the strictness of the first-order query hierarchy and a correspondence with the polynomial-time hierarchy of Stockmeyer [21]. The suggestions of Zloof [25] and of Aho and Ullman [2] concerning transitive closure transcend the first-order query hierarchy, but are easily seen to be embedded in the full hierarchy, as do those of Kowalski [17] (see also [8]). The full hierarchy to level ω^2 is then shown to have natural closure properties. The question of strictness of the higher levels, which was left open in a preliminary version of this paper [7], has been recently solved by Immerman [16]. It is shown in [16] that a single fixpoint suffices to obtain the entire hierarchy. We do, however, define a more refined collection of hierarchies characterized by the rank of the fixpoint relations allowed, and pose some open questions of strictness and definability concerning them. It is possible, however, to step out of the fixpoint hierarchy without using the full power of CQ: the fixpoint hierarchy is shown to be strictly included in the set of queries expressible in second-order predicate calculus, and the latter is strictly included in CQ.

The second part of the paper deals with sets of *higher level* queries, such as the second-order definable queries SO and the set RQ of queries computable using only ranked relation variables. The approach used in comparing such sets is to study complexity classes of queries such as QPTIME, QPHIER (queries computable in polynomial time, in the polynomial-time hierarchy), etc.

First, the problem of enumerating the queries in such classes is discussed and enumerations are shown to exist for QPHIER, QPSPACE, QEXPTIME, and classes with greater space or time resources. It is interesting that QPTIME is not known to have an enumeration. Next, it is shown that (for reasonable complexity classes) the query classes are ordered in much the same way as their corresponding complexity classes. As far as the above sets of high level queries are concerned, the main results are that $\mathrm{FP} \subset\neq \mathrm{QPTIME} \subset \mathrm{QPHIER} = \mathrm{SO} \subset\neq \mathrm{CQ}$ and $\mathrm{FP} \subset \mathrm{RQ} \subset\neq \mathrm{QPSPACE}$.

Also, QPTIME $\not\subset$ RQ, but RQ $\subset$ QPTIME iff PTIME = PSPACE, which is generally believed to be false, so that RQ is probably incompatable with QPTIME.

In the closing section we suggest areas for further research, and at the end of the paper is a list of symbols used therein.

2. Data Bases, Queries, and Operations on Queries

We first recall some basic definitions, taken essentially from [6].

Definition. Let U be some countable, *universal domain.* A *relational data base,* or *data base* for short, is a tuple $B = (D, R_1, ..., R_k)$, where $D \subset U$ is finite and for each $1 \leqslant i \leqslant k$, $R_i \subset D^{a_i}$ for some $a_i \geqslant 0$. The integer a_i is called the *rank* of R_i and B is said to be of *type* $\bar{a} = (a_1, ..., a_k)$. We shall frequently abbreviate the vector $R_1, ..., R_k$ by $\bar{R}$ and write $B = (D, \bar{R})$.

Definition. A *computable query of type* $\bar{a} \to b$ *is a partial function*

$$Q: \{B \mid B \text{ is of type } \bar{a}\} \multimap 2^{U^b}$$

satisfying the following conditions:

(i) if $B = (D, R_1, ..., R_k)$ and $Q(B)$ is defined then $Q(B) \subset D^b$,

(ii) Q is partial recursive,

(iii) if $B \xleftarrow{h} \to B'$ (i.e., B, B' are isomorphic: if $B = (D, R_1, ..., R_k)$, $B' = (D', R_1', ..., R_k')$, then h is a one–one onto function from D to D', and for all i, $R_i' = h(R_i)$), then $Q(B') = h(Q(B))$.

Conditions (i), (ii) say that Q is a partial recursive (i.e., computable) function mapping data bases into relations on the domain of the given data base. Condition (iii) is called the *consistency criterion.* It says that the output of a query should not depend on the internal representation of the data base; rather, it should treat relations of the data base as *sets* of tuples, where the representation used for elements of the tuples is unimportant. The standard queries used in the literture, such as first-order queries [4], conjunctive queries [9], tableaux queries [1], queries with transitive closure or fixpoint operators [2, 17, 22], etc. are all computable queries, and [6] gives a query language that computes exactly the computable queries. Note that in [6], queries were typed only by their inputs (i.e., a query was of type $\bar{a}$, not $\bar{a} \to b$); here, however, it is technically convenient to type with respect to output too.

Definition. Let CQ denote the class of computable queries. In the sequel, we shall sometimes omit the adjective *computable,* and simply refer to computable queries as *queries.*

We now define two fundamental operations on queries, together with the operations on sets of queries which they induce. These will serve as central concepts for the rest of the paper.

DEFINITION. Let Q be a query of type $\bar{a} \to b$. The *complement* of Q is the query $\neg Q$ of the same type, defined by,

$$\neg Q(D, \bar{R}) = D^b - Q(D, \bar{R}).$$

Also, $\neg Q(D, \bar{R})$ is undefined whenever $Q(D, \bar{R})$ is undefined. For a set of queries S, let

$$\neg S = \{\neg Q \mid Q \in S\}.$$

DEFINITION. Let $\bar{Q} = (Q_1, \ldots, Q_n)$ be a vector of queries of types $\bar{a} \to b_1, \ldots, \bar{a} \to b_n$, where $\bar{a} = (a_1, \ldots, a_k)$. If Q is a query of type $\bar{b} = (b_1, \ldots, b_n) \to c$, then the *composition* $Q \circ \bar{Q}$ is a query of type $\bar{a} \to c$, defined by

$$(Q \circ \bar{Q})(D, \bar{R}) = Q(D, \bar{Q}(D, \bar{R})) = Q(D, Q_1(D, \bar{R}), \ldots, Q_n(D, \bar{R})).$$

Again, $Q \circ \bar{Q}$ is undefined whenever one of its components is undefined. If S_1 and S_2 are sets of queries, then

$$S_1 \circ S_2 = \{Q \circ \bar{Q} \mid Q \in S_1 \text{ and if } \bar{Q} = (Q_1, \ldots, Q_n), \text{ then } Q_i \in S_2,\ 1 \leqslant i \leqslant n\}.$$

LEMMA 2.1. *For any sets of queries* $S, S_1, S_2, \ldots,$

(i) $\neg \neg S = S$,

(ii) $\neg(S_1 \circ S_2) = \neg S_1 \circ S_2$,

(iii) $(S \circ S_1) \circ S_2 = S \circ (S_1 \circ S_2)$,

(iv) $\bigcup_j (S_j \circ S) = (\bigcup_j S_j) \circ S$,

(v) $\bigcup_j (S \circ S_j) \subset S \circ (\bigcup_j S_j)$.

Proof. We prove (ii), the others are similar. For any $Q' \in \neg(S_1 \circ S_2)$ there are queries $Q \in S_1$ and $\bar{Q} = (Q_1, \ldots, Q_n)$, $Q_i \in S_2$, such that $Q'(B) = (\neg(Q \circ \bar{Q}))(B)$. But then, if $B = (D, \bar{R})$,

$$\begin{aligned} Q'(B) &= D^j - (Q \circ \bar{Q})(B) = D^j - Q(D, \bar{Q}(B)) = \neg Q(D, \bar{Q}(B)) \\ &= (\neg Q \circ \bar{Q})(B) \in \neg S_1 \circ S_2. \end{aligned}$$

Showing that any $Q' \in \neg S_1 \circ S_2$ is also in $\neg(S_1 \circ S_2)$ is analogous. ∎

In Lemma 2.1(v), the inequality cannot be changed to an equality in general. The proof of Lemma 2.2 is left to the reader.

LEMMA 2.2. *The class* CQ *of all computable queries is closed under complementation and composition.*

The operators $\neg$ and $\circ$ are two of the three fundamental operators on queries which will be considered in this paper. The third, the least fixpoint operator, will be introduced in Section 4. These operators will serve mainly to enable us to define classes of queries in a structured, algebraic manner. As a first step in this direction we now provide an algebraic characterization of the first-order definable queries.

3. THE FIRST-ORDER QUERY HIERARCHY

We have not said anything yet about the languages in which queries are formulated. Indeed, most of the sets of queries mentioned in the introduction were originally defined as the queries expressible in some query language. In this paper we shall be interested in various sets of queries, but in most cases they will be defined syntactically, i.e., in the context of some query language.

DEFINITION. Let L be the first-order language with no function symbols and with $=$ (equality) and $R_1, R_2, \ldots$ as its predicate symbols. Note that we shall be using R_i both as the formal symbol denoting a relation and as the relation itself; also, the rank of R_i will be implicit from the context, though superscripts could have been used to be completely formal. Let *First* be the language consisting of all expressions of the form

$$\bar{x}.\bar{R}.\Phi,$$

where Φ is a formula of L, $\bar{x}$ is a vector of distinct variables, containing all and only the variables appearing free in Φ, and $\bar{R}$ is a vector of distinct predicate symbols containing all those appearing in Φ.

An expression $\bar{x}.(R_1,\ldots,R_k).\Phi(\bar{x})$ in *First* represents a query Q of type $(a_1,\ldots,a_k)\to b$, where $|\bar{x}| = b$ and R_i has rank a_i. The query Q is defined by $Q(D,\bar{R}) = \{\bar{d}\in D^b \mid \Phi(\bar{d}) \text{ is true in } (D,\bar{R})\}$. For example, the expression

$$(x, y).(R_1, R_2).(\exists z)(R_1(x, z) \wedge R_2(z, y)),$$

represents the query of type $(2, 2)\to 2$ which returns the relational composition of its two arguments. Similarly the expression

$$(x, y).(R_1, R_2, R_3).(\exists z)(R_1(x, z) \wedge R_2(z, y)),$$

where the rank of R_3 is a, represents the query of type $(2, 2, a)\to 2$ returning the composition of its first two arguments. It may also be noted tht negation in *First* corresponds to complementation with respect to the domain D of the input data base.

(If $\bar{x}$ is the empty vector, then $\bar{x}.\bar{R}.\Phi$ is either the set containing the empty vector or the empty set, depending, respectively, upon whether Φ is true or false.)

DEFINITION (First-order queries). For $A \in \mathit{First}$ let Q_A denote the query represented by A, and for $\Gamma \subset \mathit{First}$ let $Q_\Gamma = \{Q_A \mid A \in \Gamma\}$. The set Q_{First} is the set of *first-order queries*, and is denoted by F.

Of course, $F \subset CQ$, but (see [2]) $F \neq \mathrm{CQ}$.

DEFINITION. For an expression A of the form $\bar{x}.\bar{R}.\Phi$ in *First*, define its negation $\neg A$ to be of the form $\bar{x}.\bar{R}.\neg\Phi$; and for $\Gamma \subset \mathit{First}$, define $\neg\Gamma = \{\neg A \mid A \in \Gamma\}$.

The following is easily verified, establishing negation of the representation of a query as the syntactic analogue of the complementation of that query.

LEMMA 3.1. *For any* $A \in \mathit{First}$, $Q_{\neg A} = \neg Q_A$, *and for any* $\Gamma \subset \mathit{First}$, $Q_{\neg\Gamma} = \neg Q_\Gamma$.

Similarly, we define *substitution*, the syntactic analogue of composition of queries.

DEFINITION. Let $A = \bar{x}.(T_1,\ldots,T_n).\Phi$, where the rank of T_i is a_i. Let $\bar{C} = (C_1,\ldots,C_n)$, where $C_i = \bar{y}_i.(R_1,\ldots,R_k).\Psi_i$, and $|\bar{y}_i| = a_i$. We write $A \circ \bar{C}$ to denote the expression $\bar{x}.(R_1,\ldots,R_k).\Phi'$, where Φ' is obtained by simultaneous substitution of the Ψ_i for the T_i in Φ, matching the $\bar{y}_i$ to the arguments of the occurrences of T_i by appropriately renaming variables which become bound or equal (note that $\bar{R}$ and $\bar{T}$ need not be disjoint).

EXAMPLE. Let $A = x.(R_1, R_2).(\exists y)(R_1(x, y, x) \wedge R_2(x, y))$, $C_1 = (x, z, y).(R_1, T_1).(\forall w)(R_1(x, y) \vee \neg T_1(w, y, z))$, $C_2 = (u, v).(R_1, T_1).R_1(u, u) \supset (\exists y)\ R_1(v, y)$. Then $A \circ (C_1, C_2) = x.(R_1, T_1).(\exists y)((\forall w)(R_1(x, x) \vee \neg T_1(w, x, y)) \wedge (R_1(x, x) \supset (\exists z)\ R_1(y, z)))$.

DEFINITION. For $\Gamma, \Delta \subset \mathit{First}$, define $\Gamma \circ \Delta = \{A \circ (C_1,\ldots,C_n) \mid A \in \Gamma,\ C_i \in \Delta,\ 1 \leqslant i \leqslant n\}$.

The following is also easily established.

LEMMA 3.2. *For any* $A, C_1,\ldots,C_n \in \mathit{First}$, $Q_{A \circ (C_1,\ldots,C_n)} = Q_A \circ (Q_{C_1},\ldots,Q_{C_n})$. *Likewise, for any* $\Gamma, \Delta \subset \mathit{First}$, $Q_{\Gamma \circ \Delta} = Q_\Gamma \circ Q_\Delta$.

DEFINITION. Let *Exist* be the set of expressions of the form

$$\bar{x}.\bar{R}.(\exists \bar{y})\,\Phi,$$

where Φ is quantifier-free. Let $E = Q_{\mathit{Exist}}$ be the set of *existential queries*.

We note that the set of conjunctive queries of [9] (and, likewise the set of tableau queries of [1]) is a subset of E, being representable by the special form

$$\bar{x}.\bar{R}.(\exists \bar{y}) \bigwedge_j R_{i_j}(\bar{u}_j),$$

where $\bar{u}_j$ is a vector of variables from $\bar{x}, \bar{y}$.

LEMMA 3.3. *For any set S of queries,*

$$S \cup \neg S \subset E \circ S = E \circ \neg S = E \circ (S \cup \neg S).$$

We are now ready to define a hierarchy of sets of expressions, which induces a hierarchy of sets of queries, similar in structure to such known hierarchies as the arithmetical and analytical hierarchies [20], and the polynomial-time hierarchy [21].

DEFINITION. Define the collection of sets of expressions of *First* $\{\Sigma_i, \Pi_i\}_{i<\omega}$, as follows:

(i) $\Sigma_0 = \{\bar{x}.\bar{R}.\Phi \mid \Phi \text{ quantifier-free}\}$,

(ii) $\Sigma_{i+1} = Exist \circ \Pi_i$,

(iii) $\Pi_i = \neg\Sigma_i$.

We are interested, not so much in the sets of formulas Σ_i, Π_i, but rather in the corresponding sets of queries denoted by Σ_i^Q, Π_i^Q:

DEFINITION. The collection of sets of queries $\{\Sigma_i^Q, \Pi_i^Q\}_{i<\omega}$, where $\Sigma_i^Q = Q_{\Sigma_i}$ and $\Pi_i^Q = Q_{\Pi_i}$ is called the *first-order query hierarchy.*

From Lemmas 3.1–3.3 we have

LEMMA 3.4. (i) $\Pi_i = \neg\Sigma_i^Q$, (ii) $\Sigma_{i+1}^Q = E \circ \Pi_i^Q = E \circ \Sigma_i^Q$, (iii) $\Sigma_i^Q \cup \Pi_i^Q \subset \Sigma_{i+1}^Q \cap \Pi_{i+1}^Q$.

The next lemma characterizes the classes Σ_i^Q (resp. Π_i^Q) as being represented by first-order queries in prenex normal form with i alternations of quantifiers beginning with existential (resp. universal) quantifiers.

LEMMA 3.5. *Let* $\exists_0$ $(= \forall_0)$ *be the set of quantifer-free formulae of the first-order language L, and let* $\exists_{i+1}$ *(resp.* $\forall_{i+1}$*) be the set of formulae of the form* $(\exists \bar{x})\Phi$ *(resp.* $(\forall \bar{x})\Phi$*) where* Φ *is in* $\forall_i$ *(resp.* $\exists_i$*) and* $\bar{x}$ *free in* Φ*. Then:*

(i) *For* Φ *in* $\exists_i$ *(resp.* $\forall_i$*),* $\bar{x}.\bar{R}.\Phi$ *(where variables in* $\bar{x}$ *are not bound in* Φ*) is in* Σ_i *(resp.* Π_i*),*

(ii) *Any query in* Σ_i^Q *(resp.* Π_i^Q*) can be represented by an expression of the form* $\bar{x}.\bar{R}.\Phi$*, where* Φ *is in* $\exists_i$ *(resp.* $\forall_i$*).*

Proof. The proof of (i) is by induction on i. For $i = 0$ it is true by definition. Let $\Phi \in \exists_{i+1}$ be of the form $(\exists \bar{x}_1)(\forall \bar{x}_2) \cdots (\Theta \bar{x}_{i+1}) \Psi$. Let A be the expression $\bar{x}, \bar{x}_1.\bar{R}.$ $(\forall \bar{x}_2) \cdots (\Theta \bar{x}_{i+1}) \Psi$. By the induction hypothesis, A is in Π_i. Then, let C be the expression $\bar{x}.T.(\exists \bar{x}_1)\, T(\bar{x}, \bar{x}_1)$. Then C is in *Exist*, and $C \circ A$ is $\bar{x}.\bar{R}.\Phi$, and by the definition, it is in Σ_{i+1} (the proof that $\forall_i$ formulae yield expressions in Π_i is analogous).

(ii) The proof is again by induction on i, and the case $i = 0$ holds by the definition. Let Q be a query in Σ^Q_{i+1} represented by $C \circ (A_1, \ldots, A_n)$, where C is in *Exist* and has the form $\bar{x}.\bar{R}.(\exists \bar{x}_1)\, \Phi(\bar{R})$, and A_j is in Π_i having the form $\bar{y}.\bar{T}.(\forall \bar{y}_{j,1})$ $(\exists \bar{y}_{j,2}) \cdots (\Theta \bar{y}_{j,i})\, \Psi_j$. Here $\bar{R} = (R_1, \ldots, R_n)$, and Φ and the Ψ_j are quantifier-free. Then substituting, for every j, $(\forall \bar{y}_{j,1}) \cdots (\Theta \bar{y}_{j,i})\, \Psi_j$ for R_j in Φ, moving negations to the right of quantifiers, and converting to prenex normal form (by first moving all leading existential quantifiers to the front, then all leading universal quantifers, etc.) results in an expression for Q of the form $\bar{x}.\bar{R}.\Phi$, where Φ is in $\exists_i$. ∎

Trivial consequences of this lemma are that $\Sigma^Q_i = E$, and for $i \geqslant 1$, $\Sigma^Q_{i+1} = \Sigma^Q_i \circ E$, $\Pi^Q_{i+1} = \Pi^Q_i \circ E$, and $\Sigma^Q_i = E \circ E \circ E \circ \cdots \circ E$ (i times). We could also have defined the first-order query hierarchy by simply taking projections on the sets defined at the previous level. Thus, if we let P denote the set of *projection queries* represented by expressions of the form

$$\bar{x}.R.(\exists \bar{y})\, R(\bar{x}, \bar{y}),$$

then we have

LEMMA 3.6. $\Sigma^Q_{i+1} = P \circ \Pi^Q_i$.

It may be noted that for $i \geqslant 1$, $P \circ \Sigma^Q_i = \Sigma^Q_i$. Another consequence of Lemma 3.5 is that the first-order Q-hierarchy describes precisely the set of first-order queries.

THEOREM 3.7. $\bigcup_i \Sigma^Q_i = \bigcup_i \Pi^Q_i = F$.

Since composition can be thought of as a way of executing a query and saving the answer for future use, any query language which can express the existential queries and also store the result of queries on the data base has the power to compute any first-order query. This is, in fact, the way by which *Query by Example* computes all first-order queries [24] and is therefore said to be complete in the sense of Codd [4].

The next result indicates some connection between the first-order query hierarchy and the polynomial-time hierarchy $\{\Sigma^P_i, \Pi^P_i\}$ in complexity theory [21]. Given a data base, it indicates the complexity of answering a query in Σ^P_i as a function of the length of the representation of the query.

DEFINITION. $B = (D, R_1, \ldots, R_k)$ is *trivial* if, for each i, either R_i is empty, or $R_i = D^{a_i}$ (i.e., R_i is *full*). Otherwise, the data base is *nontrivial*.

THEOREM 3.8. *For any i and nontrivial data base B, the set $\{\bar{d}, A \mid A \in \Sigma_i$ and $\bar{d} \in Q_A(B)\}$ is complete in Σ_i^P.*

Proof. Let C denote $\{\bar{d}, A \mid A \in \Sigma_i$ and $\bar{d} \in Q_A(B)\}$.

(i) Show $C \in \Sigma_i^P$. As in the proof of Lemma 3.5(ii), any $A \in \Sigma_i$ can be converted into the equivalent form $\bar{x}.\bar{R}.(\exists \bar{x}_{1A})(\forall \bar{x}_{2A}) \cdots (\Theta \bar{x}_{iA})\ \Phi(\bar{x}, \bar{x}_{1A}, ...)$, where Φ is quantifier-free, and this conversion does not increase the number of symbols in A. Then $(\bar{d}, A) \in C$ iff (for some appropriate polynomial poly):

$$(\exists u_1.\ \|u_1\| \leqslant \text{poly}(\|(\bar{d}, A)\|))$$
$$(\forall u_2.\ \|u_2\| \leqslant \text{poly}(\|(\bar{d}, A)\|)) \cdots (\Theta u_i.\ \|u_i\| \leqslant \text{poly}(\|(\bar{d}, A)\|))$$
$$\bigwedge_j (u_j \text{ encodes a vector } \bar{d}_{jA}) \wedge \Phi(\bar{d}, \bar{d}_{1A}, ..., \bar{d}_{iA})$$

holds. Here the u_j's are bit strings encoding vectors over the domain D of B, $\|u_j\|$, $\|(\bar{d}, A)\|$ denote the length (in bits) of u_i, $(\bar{d}, A)$. This shows that $C \in \Sigma_i^P$ since the matrix of the formula is computable in time polynomial in $\|(\bar{d}, A)\|$.

(ii) Show C is complete in Σ_i^P by reducing quantified Boolean formulae [21] to C: let T be a relation (of rank a) in $B = (D, \bar{R})$ such that $T \neq \{\ \}$, and $T \neq D_a$. Given a quantified Boolean formula Ψ of the form

$$(\exists P_{1,1}, P_{1,2}, ..., P_{1,r_1})(\forall P_{2,1}, ..., P_{2,r_2}) \cdots (\Theta P_{i,1}, ..., P_{i,r_i})\ \Phi(P_{1,1}, ..., P_{i,r_i}),$$

where the $P_{i,j}$'s are propositional symbols, it is not hard to show that Ψ is true iff $((\ ), A) \in C$, where A has the form (with each $\bar{x}_{i,j}$ being a vector of a variables),

$$(\ ).\bar{R}.(\exists \bar{x}_{1,1}, \bar{x}_{1,2}, ..., \bar{x}_{1,r_1}) \cdots (\Theta \bar{x}_{i,1}, ..., \bar{x}_{i,r_i})\ \Phi(T(\bar{x}_{1,1}), ..., T(\bar{x}_{i,r_i}))$$

(since by Lemma 3.5, $A \in \Sigma_i$). Then the result follows from the completeness for quantified Boolean formulae. ∎

It may be noted that for any trivial data base B, and any i, the set $\{\bar{d}, A \mid A \in \Sigma_i$ and $\bar{d} \in Q_A(B)\}$ is computable in polynomial time.

We can also show the strictness of the first-order query hierarchy, although it is not known whether or not the polynomial-time hierarchy is strict.

LEMMA 3.9. *For any i, $\Sigma_i^Q \subset\neq \Sigma_{i+1}^Q$.*

Proof. Let A be the expression

$$(\ ).\text{Start, Move, Win}.(\exists x_0)(\exists x_1)(\forall x_2)(\exists x_3) \cdots (\Theta x_{i+1})$$
$$\text{Start}(x_0) \wedge \text{Move}(x_0, x_1) \wedge \text{Move}(x_1, x_2) \wedge \cdots \wedge \text{Move}(x_i, x_{i+1}) \rightarrow \text{Win}(x_{i+1}).$$

The formula essentially states that the first player in an $i+1$ half-move game has a winning strategy. The domain elements correspond to game positions, Move

corresponds to the next-move relation, and Win determines if the position is a win for the first player. Now by Lemma 3.5, the corresponding query Q_A is in Σ^Q_{i+1}. We show by the technique of Ehrenfeucht–Fraïssé games [10, 13] that $Q_A \notin \Sigma^Q_i$ by showing that if the expression

$$C = (\ ).\text{Start, Move, Win.}(\exists \bar{y}_1)(\forall \bar{y}_2) \cdots (\Theta' \bar{y}_i)\Phi$$

represents a query Q_C in Σ^Q_i, then there are data bases B, B' such that $Q_A(B) = \{(\ )\}$, $Q_A(B') = \{\ \}$, but $Q_C(B) = Q_C(B')$. Note that $\{(\ )\} = D^0$ and $\{\ \} = D^0 - D^0 \neq D^0$. Let k be the maximum number of variables in any $\bar{y}_j$.

We first consider the case where i is odd. We shall define data bases B_j and B'_j inductively as follows. $B_0 = (D_0, \text{Start}_0, \text{Move}_0, \text{Win}_0)$ and $B'_0 = (D_0, \text{Start}_0, \text{Move}_0, \text{Win}'_0)$, where $D_0 = \{d\}$, $\text{Start}_0 = \text{Win}_0 = \{(d)\}$, $\text{Move}_0 = \text{Win}'_0 = \{\ \}$. For any j, let $B_j = (D_j, \text{Start}_j, \text{Move}_j, \text{Win}_j)$, $B'_j = (D_j, \text{Start}_j, \text{Move}_j, \text{Win}'_j)$, with $D_j = \{d_1, d_2, ..., d_n\}$, and $\text{Start}_j = \{d_1\}$. Then $B_{j+1} = (D_{j+1}, \text{Start}_{j+1}, \text{Move}_{j+1}, \text{Win}_{j+1})$ and $B'_{j+1} = (D_{j+1}, \text{Start}_{j+1}, \text{Move}_{j+1}, \text{Win}'_{j+1})$, where $D_{j+1} = \{d_0, d_{1,1}, d_{1,2}, ..., d_{1,n}, ..., d_{k+1,n}\}$, $\text{Start}_{j+1} = \{d_0\}$, $\text{Move}_{j+1} = \{(d_0, d_{p,1}) \mid 1 \leqslant p \leqslant k+1\} \cup \{(d_{p,q}, d_{p,r}) \mid 1 \leqslant p \leqslant k+1,\ (d_q, d_r) \in \text{Move}_j\}$, $\text{Win}_{j+1} = \{d_{1,q} \mid d_q \in \text{Win}'_j\} \cup \{d_{p,q} \mid 2 \leqslant p \leqslant k+1, d_q \in \text{Win}_j\}$, and $\text{Win}'_{j+1} = \{d_{p,q} \mid 1 \leqslant p \leqslant k+1, d_q \in \text{Win}_j\}$. Now let B be B_{i+1}, and B' be B'_{i+1}. The B_j, B'_j can be thought of as game trees defined inductively as in Fig. 1, and it is not hard to see that the expression A is true for B_{i+1} (by choosing x_1 to be $d_{1,1}$; recall that i is odd) but false for B'_{i+1}.

We now play an Ehrenfeucht–Fraïssé game on B, B' as follows: The first player tries to demonstrate that B, B' are not isomorphic, and the second player tries to prevent him from doing so. The first player chooses any sequence of k (not necessarily distinct) elements from the domain of either B or B' (the domains can be thought of as being tagged to make them distinct). Let us say he picks k elements from B. The second player chooses a corresponding sequence of k elements from the other data base, i.e. from B'. This completes one move. Now the first player chooses k elements from B', and the second player chooses some corresponding k elements from B. This completes the second move. Next, the first player again chooses k elements from B, and so on for a total of i moves. At that point, let $d_1, d_2, ..., d_{ik}$ be the elements chosen by the two players from B, and $d'_1, d'_2, ..., d'_{ik}$ the corresponding elements from B'. Then the second player wins if $(d_1, ..., d_{ik})$ are isomorphic to $(d'_1, ..., d'_{ik})$, i.e., if for every p, $\text{Win}(d_p)$ holds in B iff $\text{Win}'(d'_p)$ holds holds in B', likewise for Start, and for every p, q, $\text{Move}(d_p, d_q)$ holds in B iff $\text{Move}'(d'_p, d'_q)$holds

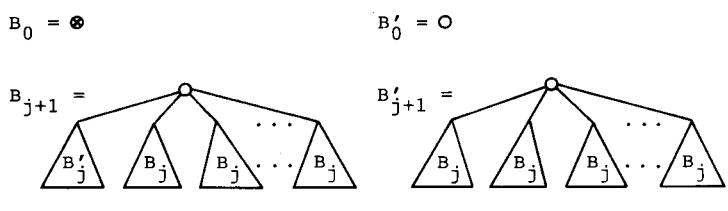

FIG. 1. ⊗, win for first player; O, not a win for first player.

in B'. Otherwise, the first player wins. From (a minor modification of) the Ehrenfeucht–Fraïssé theorem [10, 13], it follows that if the second player has a winning strategy, then $Q_C(B) = Q_C(B')$. We shall show that the second player indeed has a winning strategy. In fact, we shall show that the second player wins even an $i+1$ move game, where the first player must start choosing elements from $B' = B'_{i+1}$. The proof is by induction on the number of moves m (then $m = i+1$ gives the desired result), and the case $m = 0$ is trivial. For an m move game, $m \geqslant 1$, we shall, for convenience, rename the elements in the domain of B' from $d_0, d_{1,1}, \ldots, d_{k,n}$ to $d'_0, d'_{1,1}, \ldots, d'_{k,n}$. The second player uses the following strategy: Whenever the first player chooses d_0 or d'_0, the second player chooses the other one. Suppose the first player chooses elements $d'_{p_1,q_1}, \ldots, d'_{p_k,q_k}$. There must be some number $\leqslant k+1$ which is not included in $p_1, \ldots, p_k$. Without loss of generality, suppose for all i, $p_i \neq 1$ (can be obtained by systematic renaming of the elements in D'_m). The second player then chooses $d_{p_1,q_1}, \ldots, d_{p_k,q_k}$, and subsequently, if the first player ever chooses $d_{p,q}$ (resp. $d'_{p,q}$) $p \geqslant 2$, the second player responds with $d'_{p,q}$ (resp. $d_{p,q}$). These choices can never be used by the first player to show nonisomorphism. Therefore his subsequent moves may be confined to elements of the form $d_{1,q}, d'_{1,q}$, starting with the former. The game is then reduced to an $m-1$ move game played on B_{m-1}, B'_{m-1}, starting with the first player choosing from B'_{m-1}, which, by the induction hypothesis, is a win for the second player. This completes the induction, and the proof for odd i.

The proof for even i is similar. Data bases B_j, B'_j are defined as before, except that the inductive definition is started with $\mathrm{Win}_0 = \{\ \}$, and $\mathrm{Win}'_0 = \{d\}$. This is needed to show that with the definitions $B = B_{i+1}$ and $B' = B'_{i+1}$, $Q_A(B) = \{(\ )\}$, $Q_A(B') = \{\ \}$. The Ehrenfeucht–Fraïssé game, however, works out identically with the previous case. ∎

THEOREM 3.10. *For any* $i \geqslant 1$, $\Sigma_i^Q \neq \Pi_i^Q$.

Proof. For any $i \geqslant 1$, if $\Sigma_i^Q = \Pi_i^Q$, then $\Sigma_{i+1}^Q = P \circ \Pi_i^Q = P \circ \Sigma_i^Q = \Sigma_i^Q$ (by Lemma 3.6 and the remark following it), which contradicts Lemma 3.9. ∎

THEOREM 3.11. *For any* i, $\Sigma_i^Q \cup \Pi_i^Q \subset\neq \Sigma_{i+1}^Q \cap \Pi_{i+1}^Q$.

Proof. It suffices (from Lemma 3.4(iii)) to show "$\neq$". From Theorem 3.10, there is a query Q in Σ_i^Q which is not in Π_i^Q. Say Q is represented by $\bar{x}.\bar{R}.\Phi$. Let T be a new zero-ary predicate symbol, and consider the query Q' represented by $\bar{x}.(\bar{R}, T).(T \wedge \Phi) \vee (\neg T \wedge \neg \Phi)$. By using Lemma 3.5, $Q' \in \Pi_{i+1}^Q$, but is not in $\Sigma_i^Q \cup \Pi_i^Q$, for if say $Q' \in \Sigma_i^Q$, represented by $\bar{x}.(\bar{R}, T).\Psi(T)$, then $\neg Q$ can be represented by $\bar{x}.\bar{R}.\Psi$ (false), implying that $\neg Q \in \Sigma_i^Q$; a contradiction. The case $Q' \in \Pi_i^Q$ is similar. ∎

Theorems 3.10 and 3.11 establish the strictness of the first-order hierarchy. Keisler and Walkoe [18] have proved similar looking results establishing that no two quantifier prefixes of the same length have the same power of expression over finite data bases. The two results are, however, independent, since in [18] the length of the prefix

is of interest and not its type. Thus, that $\exists\exists\forall \neq \exists\forall\forall$ follows from [18] and not from the present paper, and that $\exists\forall \neq \forall\forall\forall$ follows from the present paper and not from [18].

4. Fixpoint Queries

In this section the first-order hierarchy is extended to include classes Σ_α^Q and Π_α^Q for all $\alpha < \omega^2$, rather than just $\alpha < \omega$. The idea is to obtain Σ_α^Q for a limit ordinal α by attaching a least fixpoint operator Y to queries in the sets defined for smaller ordinals, i.e. to Σ_β^Q, for $\beta < \alpha$.

Definition. Let LY be the first-order language L of Section 3, augmented with the additional formation rule: Let Φ be in LY, and let the predicate symbol R of rank a appear positively in Φ (i.e., each free occurrence of R is under an even number of negations). Then $(\bar{z}.YR(\bar{x}))\,\Phi$ is in LY, where $\bar{x}$ is an a-tuple of distinct variables and $\bar{z}$ is also an a-tuple of (not necessarily distinct) variables. Let Ψ denote $(\bar{z}.YR(\bar{x}))\,\Phi$. The variables of $\bar{x}$ are *bound* in Ψ. The free variables of Ψ are the free variables of Φ (excluding $\bar{x}$) in addition to the variables in $\bar{z}$. The free predicate symbols of Ψ are those that are free in Φ, excluding R (which is bound in Ψ). LY is, of course, closed under the logical connectives $\wedge$, $\vee$, $\neg$, and quantifiers $\exists$, $\forall$.

Definition. Let Ψ denote $(\bar{z}.YR(\bar{x}))\,\Phi(R, \bar{x}, \bar{y})$ as in the above definition, with $\bar{y}$ being a vector (without duplicates) of the free variables in Φ, other than those in $\bar{x}$. Given meanings for the predicate symbols free in Φ (excluding R), and given an assignment θ which assigns $\theta(u) \in D$ for each variable u free in Ψ (for a vector $\bar{u} = (u_1,..., u_k)$ we shall write $\theta(\bar{u})$ for $(\theta(u_1),..., \theta(u_k))$), define an a-ary relation T as follows:

(i) For every $\bar{d}$, $T(\bar{d})$ holds iff $\Phi(T, \bar{d}, \theta(\bar{y}))$ (or less formally: $T \equiv \Phi(T)$, i.e., T is a fixpoint of Φ), and

(ii) For any a-ary relation T', if for every $\bar{d}$, $T'(\bar{d})$ holds iff $\Phi(T', \bar{d}, \theta(\bar{y}))$ does, then $T \subset T'$ (i.e., T is the least fixpoint).

Then the formula Ψ is *satisfied* by the assignment θ iff $T(\theta(\bar{z}))$ holds.

An equivalent definition of fixpoints could have been obtained by combining the free variables $\bar{y}$ in the above definition with the bound variables $\bar{x}$ and writing the formulas as $(\bar{z}'.YR')\,\Phi'$. Here the length of $\bar{z}'$, the arity of R' and the number of free variables in Φ' are equal.

The next theorem shows that such a T exists, and therefore the definition is sound.

Theorem 4.1. *The relation T in the above definition exists, and satisfies $T = \bigcup_i T_i$, where $T_0 = \{\ \}$ and $T_{i+1} = \{\bar{d} \mid \Phi(T_i, \bar{d}, \theta(\bar{y}))$ holds$\}$.*

Proof. It is not hard to show that positivity implies monotonicity. Hence, since $\{\ \} \subset T_1$, $T_1 = \{\bar{d} \mid \Phi(\{\ \}, \bar{d}, \theta(\bar{y}))\} \subset \{\bar{d} \mid \Phi(T_1, \bar{d}, \theta(\bar{y}))\} = T_2$. By induction on i, $T_i \subset T_{i+1}$, for every i. Since the domain is finite, there is a j such that $\bigcup_i T_i = T_j$. Then $\bar{d} \in T_{j+1}$ iff $\Phi(T_{j+1}, \bar{d}, \theta(\bar{y}))$ holds. Thus T_j is a fixpoint of Φ in the sense of clause (i) of the definition. Now to show that $\bigcup_i T_i = T_j$ is the least such (clause (ii)), let T' satisfy the equivalence $T'(\bar{d})$ iff $\Phi(T', \bar{d}, \theta(\bar{y}))$. We show that $T_i \subset T'$ by induction on i. Clearly, $T_0 = \{\ \} \subset T'$. Assume $T_i \subset T'$. Then $T_{i+1} = \{\bar{d} \mid \Phi(T_i, \bar{d}, \theta(\bar{y}))\} \subset \{\bar{d} \mid \Phi(T', \bar{d}, \theta(\bar{y}))\} = T'$ (by the monotonicity of Φ). ∎

This version of the Knaster–Tarski theorem illustrates the usual way of calculating least fixpoints by a sequence of approximations from below, which, in this case, is finite.

Remark. In many cases, the $\bar{z}$ in $(\bar{z}.YR(\bar{x}))\Phi$ will be simply $\bar{x}$, but formally we want to allow identification of variables as, e.g., in $((v, w, v).YR(\bar{x}))\Phi$, without having to resort to conjunctions with equality terms. Whenever there is no confusion we shall write $(\bar{z}.YR(\bar{x}))\Phi$ simply as $(YR)\Phi$.

DEFINITION. Let *Fixpoint* be the language consisting of all expressions of the form $\bar{x}.\bar{R}\Phi$, where Φ is a formula of LY, $\bar{x}$ is a vector of distinct variables containing at least those appearing free in Φ, and $\bar{R}$ is a vector of distinct predicate symbols containing at least those appearing free in Φ. By the previous definition it should be clear how to associate a query Q_A with an expression $A \in$ *Fixpoint*. In particular, if $A = (\bar{z}.YR(\bar{x}))\,\Phi(R, \bar{x}, \bar{y})$, then $Q_A(R_1, \ldots, R_n) = \{\theta(\bar{z}) \mid T(\theta(\bar{z}))$, all $\theta\}$.

DEFINITION (Fixpoint queries). For $\Gamma \subset$ *Fixpoint*, let $Q_\Gamma = \{Q_A \mid A \in \Gamma\}$. The set $Q_{Fixpoint}$ is called the set of *Fixpoint queries*, and is denoted FP.

The definitions of negation ($\neg$) and substitution ($\circ$) in Section 3 extend from the set *First* of first-order expressions to *Fixpoint*, and Lemmas 3.1, 3.2 hold with *Fixpoint* replacing *First*.

DEFINITION. Let $A = \bar{x}.(R_1, \ldots, R_k)\Phi$ be an expression in *Fixpoint*. Any expression in *Fixpoint* of the form $\bar{v}.(R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_k).\bar{z}.(YR_i)\Phi$ is called a *fixpoint of* A. The set of fixpoints of A is denoted YA. Also, for $\Gamma \subset$ *Fixpoint*, let $Y\Gamma = \bigcup_{A \in \Gamma} YA$. Thus, for any $A \in$ *Fixpoint*, $Q_{YA} \subset$ FP.

EXAMPLE. Let Φ be

$$x = y \vee (\exists z)(R(x, z) \wedge R'(z, y)).$$

Let $A = (x, y).(R, R').\Phi$ and $K = (x, y).R.((x, y).YR'(x, y))\Phi$ be expressions in *Fixpoint*. Then K is a fixpoint of A, and the set Q_{YA} includes the reflexive transitive closure query Q_K denoted by TC. Since $A \in \Sigma_1$, we have $\mathrm{TC} \in Y\Sigma_1^Q$. Thus the reflexive transitive closure can be described as a fixpoint of an existential query.

For a set S of queries such as F, E, FP, etc., with which an underlying set Γ of expressions is clearly associated, we shall loosely use the notation YS to stand for $Q_{Y\Gamma}$. For example, YF is the set of queries which are represented by fixpoints of first-order expressions.

With this convention in mind, FP is the closure of E (and, for that matter, of F) under $\circ$ and Y. It is also the closure of C (the set of conjunctive queries) under $\neg$, $\circ$, and Y.

We now define the fixpoint hierarchy.

DEFINITION. Define the collection of sets of expressions of $Fixpoint\{\Sigma_\alpha, \Pi_\alpha\}_{\alpha<\omega^2}$ as follows:

(i) $\Sigma_0 = \{\bar{x}.\bar{R}.\Phi \mid \Phi \text{ quantifier-free}\}$,

(ii) $\Sigma_{\alpha+1} = Exist \circ \Pi_\alpha$, for any α,

(iii) $\Sigma_\alpha = Y(\bigcup_{\beta<\alpha} \Sigma_\beta)$ for a limit ordinal α, and

(iv) $\Pi_\alpha = \neg\Sigma_\alpha$, for any α.

DEFINITION. The collection of sets of queries $\{\Sigma^Q_\alpha, \Pi^Q_\alpha\}_{\alpha<\omega^2}$, where $\Sigma^Q_\alpha = Q_{\Sigma_\alpha}$ and $\Pi^Q_\alpha = \Pi_{Q_\alpha}$ is called the *Fixpoint query hierarchy*, or *Fixpoint hierarchy*, for short.

We justify our use of a single application of Y in clause (iii).

LEMMA 4.2. *For any* $i > 0$, $Y\Sigma^Q_{\omega\cdot i} = \Sigma^Q_{\omega\cdot i}$.

(Note that $\omega \cdot i$ is i times the ordinal ω; i.e., the ith application of a fixpoint.)

Proof. The proof of this lemma is a straightforward generalization of a result appearing in [8] to the effect that mutual and double fixpoints can be collapsed into single fixpoints. ∎

Table I illustrates the Σ^Q_α classes. The notation in the table is justified by the following generalization of Theorem 3.7:

THEOREM 4.3. *For every* k, $\bigcup_j \Sigma^Q_{\omega\cdot k+j} = F \dot{\circ} \Sigma^Q_{\omega\cdot k}$.

TABLE I

β	Σ^Q_β	$\Sigma^Q_{\beta+1}$	$\Sigma^Q_{\beta+2}$	$\cdots \Sigma^Q_{\beta+i} \cdots$	$\bigcup_i \Sigma^Q_{\beta+i}$
0	Σ^Q_0	E	$E \circ E$	$\cdots E^i \cdots$	F
$\omega \cdot 1$	YF	$E \circ YF$	$E \circ E \circ YF$	$\cdots E^i \circ YF \cdots$	$F \circ YF$
$\omega \cdot 2$	$Y(F \circ YF)$	$E \circ Y(F \circ YF)$	$E \circ E \circ Y(F \circ YF)$	$\cdots E^i \circ Y(F \circ YF) \cdots$	$F \circ Y(F \circ YF)$
⋮	⋮	⋮	⋮	⋮	⋮

Proof. $\bigcup_j \Sigma^Q_{\omega\cdot k+j} = \bigcup_j \Sigma^Q_j \circ \Sigma^Q_{\omega\cdot k} = (\bigcup_j \Sigma^Q_j) \circ \Sigma^Q_{\omega\cdot k} = F \circ \Sigma^Q_{\omega\cdot k}$. ∎

The following too is straightforward:

THEOREM 4.4. $\bigcup_{\alpha<\omega^2} \Sigma^Q_\alpha = \mathrm{FP}$.

Since FP is closed under the operations from which the hierarchy is constructed, and since we know that the union of the hierarchy is FP, it is obvious that an attempt to extend the definition to ordinals higher than ω^2 does not result in any new queries.

Most of the queries considered in the literature are associated with sets in the Q hierarchy. The transitive closure query TC is clearly in $\Sigma^Q_\omega = YF$ by Example 4.1. Also, [2] suggested extending Codd's relational algebra [4] by allowing least fixpoints to be applied to expressions in the algebra. Since the set of queries expressible in the relational algebra is precisely F (the set of queries expressible in the first-order relational calculus), it appears that the language suggested in [2] is $F \circ YF$, i.e. contained in $\Sigma^Q_{\omega\cdot 2}$.

It is possible to view the language of Horn-clause logic programs as a query language (see e.g., [15, 17, 22]). It can be shown that the queries definable in this language correspond to YE^+, where E^+ is the set of queries represented by positive existential first-order formulae. Thus Horn queries are strictly contained in YF. See [8].

An interesting question concerns the strictness of the fixpoint hierarchy, or, to be more precise, the possibility of generalizing Theorems 3.10 and 3.11.

To this end, in response to the appearance of this question in a preliminary version of this paper [7], Immerman [16] has shown that in fact $\mathrm{FP} = \Sigma^Q_{\omega+1}$, i.e., that a single fixpoint suffices to obtain the entire hierarchy. This surprising result is obtained by first showing how to express negations of fixpoints as fixpoints of wider degree. Multiple and nested fixpoints are then collapsed into single ones by additional widening, as in [8]. In particular, Immerman's result shows how to encode $\neg TC$ in YF by a fixpoint of a 7-ary relation.

It is of some interest, expecially in view of [16], to investigate the fixpoint hierarchies obtained by restricting the width of bound relation symbols.

DEFINITION. For $\Gamma \subset \mathit{Fixpoint}$, let $\Gamma^{[j]}$ be those formulas in Γ where the fixpoint operator Y is applied to relations of at most rank j. $Q_{\mathit{Fixpoint}}[j]$ is denoted by $\mathrm{FP}^{[j]}$, and the appropriate hierarchies of formulas and queries by $\Sigma^{[j]}_\alpha$, $\Pi^{[j]}_\alpha$ and $\Sigma^{Q[j]}_\alpha$, $\Pi^{Q[j]}_\alpha$, respectively.

Clearly, the first ω steps of each restricted hierarchy are the same, since they contain no fixpoints. Also, it is easy to see that for each $j > 1$, $YF^{[j]} \neq F$, since $\mathrm{TC} \notin F$ ([2]), but $\mathrm{TC} \in \Sigma^{Q[j]}_\omega$. Now, although by [16] $\neg\mathrm{TC} \in YF$, we can show the following:

THEOREM 4.5. $\neg$TC *is not in* $YF^{[2]}$.

Proof. We show that a contradiction results from assuming that $\neg TC \in YF^{[2]}$. Let $\neg TC$ be representable by the expression

$$A = (x, y).R.(YR'')\,\Phi(R, R'', \bar{z}),$$

where Φ is a first-order formula, R'' is binary and $\bar{z}$ is a sequence of the free variables in Φ, which may contain x, y. Consider the data bases $B_n = (D_n, R_n)$, $B'_n = (D'_n, R'_n)$, where $D_n = \{d_1,..., d_{2n}\}$, $D'_n = \{d'_1,..., d'_{2n}\}$, $R_n = \{(d_1, d_2), (d_2, d_3),..., (d_{2n-1}, d_{2n}), (d_{2n}, d_1)\}$, and $R'_n = \{(d'_1, d'_2),..., (d'_{n-1}, d'_n), (d'_n, d'_1)\} \cup \{(d'_{n+1}, d'_{n+2}),..., (d'_{2n-1}, d'_{2n}), (d'_{2n}, d'_{n+1})\}$. Essentially, B_n consists of a cycle of length $2n$, B'_n consists of two cycles, each of length n. Now $\neg TC(B_n) = Q_A(B_n) = \{\ \}$; hence given any assignment θ to the variables $\bar{z}$, using Theorem 4.1, $\Phi(R_n, \{\ \}, \theta(\bar{z}))$ must be false.[1] Hence, $\exists \bar{z} \Phi(R, \{\ \}, \bar{z})$ is false in B_n. We shall show by using Ehrenfeucht–Fraïssé games that, for large enough n, the formula $\exists \bar{z} \Phi(R, \{\ \}, \bar{z})$ cannot distinguish B_n from B'_n, i.e., it is false in B'_n. But then $Q_A(B'_n) = \{\ \} \neq \neg TC(B'_n)$, which is the desired contradiction.

Consider an Ehrenfeucht–Fraïssé game played on B_n, B'_n, where each player chooses one element at a time. We show that the second player wins a k_1 move game if $n > 2^{k_1 - 1}$.

We first introduce some notation. For $e \in D_n$ (resp. D'_n) define $e + 1 = e_1$, where $R(e, e_1)$ (resp. $R'(e, e_1)$). For $e, e_1 \in D \cup D'$, define $e + 0 = e$, $e + (i + 1) = (e + i) + 1$, $e - i = e_1$ where $e_1 + i = e$, $e - e_1 = \min\{j \geqslant 0 \mid e + j = e_1\}$ (here $\min\{\ \}$ is ∞), and $\text{dist}(e, e_1) = \min\{e - e_1, e_1 - e\}$. Essentially, $e + i$ denotes the element in the cycle i steps after e, and $\text{dist}(e, e_1)$ denotes the distance between e and e_1. An *m-chain* is a sequence of elements $\bar{e} = e_1, e_2,..., e_p$ such that for all i, $e_{i+1} - e_i \leqslant m$, and $\sum_i (e_{i+1} - e_i) < n - m$. Two chains $e_1,..., e_p$, and $f_1,..., f_q$ are *m-disjoint* if for each e_i, f_j we have $\text{dist}(e_i, f_j) > m$, and they are *isomorphic* if $p = q$ and for all i, $e_{i+1} - e_i = f_{i+1} - f_i$. Essentially, a chain consists of a sequence of elements close together; chains are disjoint if they are far apart, and they are isomorphic if the spacings within the chains match. An element e is *m-free* from a chain $e_1,..., e_p$ if for all i, $\text{dist}(e, e_i) > m$.

The second player wins by the following strategy. Inductively, with k moves remaining (the induction will be on k starting from k_1 and decreasing to 0), the elements from D (resp. D') chosen by both players can be partitioned into several 2^k-chains $\bar{e}_1, \bar{e}_2,..., \bar{e}_q$ (resp. $\bar{e}'_1,..., \bar{e}'_q$) that are mutually 2^k-disjoint; and $\bar{e}_i, \bar{e}'_i$ are isomorphic for each i. Since a p-element 2^k-chain has at most $1 + p\,2^k$ elements that are not 2^{k-1} free from it, it follows from a counting argument that there will be an element in D (resp. D') which is 2^{k-1}-free from each $\bar{e}_i$ (resp. $\bar{e}'_i$) (note that the total number of distinct elements in the chains is no more than $k_1 - k$, and $n > 2^{k_1 - 1}$). The inductive hypothesis is trivially true in the beginning with no elements chosen, and k_1

[1] Note that this is so only because R'' is binary and thus each θ for which $\Phi(R_n, \{\ \}, \theta(\bar{z}))$ is true contributes towards the least fixpoint of Φ, hence towards $\neg TC(B_n)$ which should be empty. If R'' were, say, ternary it might have been possible to have $(x, x, y).R.YR''\Phi$ encode $\neg TC$ and indeed $\Phi(R, \{\ \}, \theta(\bar{z}))$ might be true, but contributing no tuples of the form (a, a, b).

moves remaining. Now, with k moves remaining, and the inductive hypothesis holding, say the first player picks an element $d \in D$ (the case when he picks an element from D' is analogous). It can be checked there are two cases: (i) d is 2^{k-1}-free from all $\bar{e}_i$, and (ii) d is not 2^{k-1}-free from exactly one $\bar{e}_i$. In case (i), the second player picks any element d' in D' which is 2^{k-1}-free from each $\bar{e}'_i$. Then d and d' form (trivially isomorphic) 2^{k-1}-chains, and the e^k-chains $\bar{e}_i, \bar{e}'_i$ partition into one or more 2^{k-1}-chains each. The resulting chains all obey the induction hypothesis for $k-1$. In case (ii), let $d = e + j$, where e is in $\bar{e}_i$ and $j \leqslant 2^{k-1}$ (the case $d = e - j$ is similar), and e' be the element in $\bar{e}'_i$ corresponding to e; then the second player chooses $e' + j$. Again it can be seen that the induction hypothesis is satisfied for $k-1$. Finally, with $k = 0$, i.e., no move remaining, from the induction hypothesis, the second player wins the game. ∎

Remark. Aho and Ullman [2] show that TC is not in F. In fact, Fagin [12] showed that a problem closely related to TC (that of expressing whether all elements of the domain are connected by R) is not expressible in second-order predicate calculus, where the second-order quantifiers are all universal and monadic (formula is in prenex form and second-order quantifiers precede first-order quantifiers).

Thus, Theorem 4.5, together with Theorem 3.10 yields

COROLLARY 4.6. *For* $\alpha \leqslant \omega$, $\Sigma_\alpha^{Q[2]} \neq \Pi_\alpha^{Q[2]}$.

We can in fact prove

THEOREM 4.7. *For each* $j > 1$, $\alpha \leqslant \omega$, $\Sigma_\alpha^{Q[j]} \neq \Pi_\alpha^{Q[j]}$.

Sketch of proof. For $\alpha < \omega$, this is precisely Theorem 3.10. To show the claim for $\alpha = \omega$, consider the j-fold transitive closure query, $\mathrm{TC}^{[j]}$ of type $(j) \to j$, defined as follows:

$$(x_1,\ldots,x_j).YS.\left(R(x_1,\ldots,x_j) \vee (\exists y)\left(\bigvee_{i=1}^{j-1} (S(x_1,\ldots,x_{j-1}[y/i]) \wedge S(x_2,\ldots,x_j[y\backslash i])\right)\right),$$

where $\bar{x}[y/i]$ and $\bar{x}[y\backslash i]$ stand, respectively, for the vectors obtained from $\bar{x}$ by inserting y after or before the ith element of $\bar{x}$. As an example, the 3-fold transitive closure of a ternary relation R is obtained by searching for 4-tuples x, y, z, w with (x, y, z) and (y, z, w) in the closure and adding (x, z, w) and (x, y, w) to the closure. Since the above definition shows that $\mathrm{TC}^{[j]} \in \Sigma_\omega^{Q[j]}$, it suffices to show $\neg\mathrm{TC}^{[j]} \notin \Sigma_\omega^{Q[j]}$. This is established just as in Theorem 4.5 by considering models $B_n^{[j]}$ and $B_n'^{[j]}$ consisting of one cycle of length $2n$ and two, each of length n, respectively, in which each j consecutive elements are related via R. The argument that for sufficiently large n a first-order formula cannot distinguish between the two models is almost identical. Details are left to the reader. ∎

We pose the following questions:

Open Question. Is $\Sigma_a^{Q[j]} \neq \Pi_a^{Q[j]}$ for all $j > 1$, $a < \omega^2$?

Clearly $\mathrm{FP}^{[j]} \subset \mathrm{FP}^{[j+1]}$ for $a > 1$:

Open Question. Is $\mathrm{FP}^{[j]} \neq \mathrm{FP}^{[j+1]}$ for all $j > 1$?

In response to [7], Gaifman [14] has shown the following:

(1) $\Sigma_{\omega+i}^{Q[2]} \neq \Pi_{\omega+i}^{Q[2]}$ for all i,

(2) $\mathrm{FP}^{[2]} \neq \mathrm{FP}^{[3]}$.

The general questions, though, remain open.

Finally, we state the following fact comparing the Fixpoint hierarchy with the second-order queries:

DEFINITION (Second-order queries). Let SO be the set of queries expressible by expressions of the form $\bar{x}.\bar{R}.\Phi$, where Φ is a second-order formula (second-order quantifiers are over relations) whose free relation symbols are from $\bar{R}$ (corresponding to the relations of a data base) and $=$ (equality), which has no function symbols, and whose free variables are from $\bar{x}$.

PROPOSITION 4.8. $\mathrm{FP} \subset\neq \mathrm{SO}$.

It is easy to see that $\mathrm{FP} \subset \mathrm{SO}$, since $YR.\Phi$ can be written in SO in the general form corresponding to $R = \Phi(R) \wedge \forall R'\ (R' = \Phi(R') \rightarrow R \subset R')$. $\mathrm{FP} \neq \mathrm{SO}$ is proved in the next section.

5. COMPLEXITY CLASSES FOR QUERIES

As mentioned in the introduction, we shall use complexity-theoretic methods to classify some powerful query languages and their associated sets of queries. We would like to define, say, the class of queries that can be computed in polynomial time (i.e., polynomial in the data base), and likewise for queries that can be computed in polynomial space, exponential time, etc. Recalling that a query is a partial recursive function from data bases to relations, we may simply restrict attention to those queries Q for which the set $\{(B, Q(B)) \mid B \text{ is a data base}\}$ is a language in the appropriate complexity class. We chose a slightly different definition.

DEFINITION. Let PTIME (resp. EXPTIME, LOGSPACE, PSPACE) be the class of languages S such that there is a polynomial $P(n)$ and a deterministic Turing machine which accepts S in time $P(n)$ (resp. time $2^{P(n)}$, space $\log(P(n))$, space $P(n)$). Here n refers to the length of the input. Let $\mathrm{PHIER} = \bigcup_i \Sigma_i^p$ be the polynomial-time hierarchy [21].

DEFINITION. If C is a class of sets (e.g., a complexity class like PTIME), then the class QC of queries in C is defined to be

$$\mathrm{QC} = \{Q \mid Q \text{ is a total computable query, and } \{(B, \bar{x}) \mid \bar{x} \in Q(B)\} \in C\}.$$

Here we assume some standard encoding of $(B, \bar{x})$ into strings. This defines the set of queries computable in polynomial time QPTIME, in polynomial space QPSPACE, etc.

Note that the above definition ignores queries that are not total. This is reasonable for all the complexity classes.

We have defined complexity classes for queries by the problem of recognizing if a tuple is in the output. It may be noted that the space/time complexity of computing the output is closely related, because if $\{(B, \bar{x}) \mid \bar{x} \in Q(B)\}$ can be recognized in time $T(n)$, $T(n) \geqslant n$ (resp. space $S(n)$, $S(n) \geqslant \log(n)$), then $Q(B)$ can be computed in time $n^k T(n)$ for some k (resp. in space $S(n)$). Thus, for example, if $Q \in$ QPTIME, then given any B, $Q(B)$ can be computed in time polynomial in the length of the standard encoding of B.

The reader may contrast this definition of the complexity of queries with Theorem 3.8 (see also [9, Sect. 4]), where the data base is fixed and the input includes the representation of the query. Following a preliminary version of this paper [7], these two types of complexities have been further investigated by Vardi [23] who has shown that in a number of instances these complexities differ by one exponential.

Enumerating the queries in a complexity class is not immediate from an enumeration of the Turing machines in that class. The difficulty is with the consistency criterion (condition (iii) in the definition of computable queries, Section 2), which requires that queries preserve isomorphisms. This condition is, of course, not decidable for an arbitrary Turing machine. One can, however, modify Turing machines so as to satisfy the consistency criterion.

DEFINITION. Given a class $C = \{C_0, C_1, \ldots\}$ of languages, the set of queries QC has an *effective enumeration* S if $S \subset \{0, 1, \ldots\}$ is total recursive and

(i) $Q\{C_i \mid i \in S\} = QC$, and

(ii) $\forall i \in S\ \exists Q \in QC$ such that $C_i = \{(B, \bar{x}) \mid \bar{x} \in Q(B)\}$.

CONSTRUCTION. Given $\bar{a}, b$, and a Turing machine M that halts on all inputs, construct a Turing machine M' as follows: On input $(B, \bar{d})$, where $B = (D, \bar{R})$ is of type $\bar{a}$ and $\bar{d} \in D^b$, M' computes the least (in some well ordering) B' isomorphic to B. Then M' accepts the input $(B, \bar{d})$ iff there is an isomorphism h mapping B to B' (write $B \leftarrow^h \rightarrow B'$) and M accepts the input $(B', h(\bar{d}))$. Let Q_M be a function from data bases to relations such that $Q_M(B) = \{\bar{d} \mid M \text{ accepts } (B, \bar{d})\}$. Let $Q_{M'}$ be similarly defined. With these definitions it is easy to show

LEMMA 5.1. (i) $Q_{M'}$ *is a computable quary, and* (ii) *if* Q_M *is a computable query of type* $\bar{a} \to b$, *then* $Q_{M'} = Q_M$.

This lemma gives us effective enumerations of QPSPACE, QEXPTIME, and classes with greater space or time resource. This does not, however, give an effective enumeration of QPTIME, because it is not known whether B' in the construction can be computed in polynomial time.

Open Question. Does QPTIME have an effective enumeration?

LEMMA 5.2. *For* $i \geqslant 2$, $Q\Sigma_i^P$, $Q\Pi_i^P$ have effective enumerations; and so does QPHIER.

Proof. Given a fixed encoding of data bases into strings, define $B < B'$ to mean either that the encoding of B is shorter than that of B', or that they have equal length and the encoding of B precedes that of B' in some fixed lexicographic ordering. Let $B \leqslant B'$ mean that $B < B'$ or $B = B'$. Given any formula $\Phi(B, \bar{x})$ of the form

$$(\exists y_1)(\forall y_2) \cdots (\Theta y_i)\, P(B, \bar{x}, y_1, \ldots, \bar{y}_i)$$

representing a set in Π_i^P, i.e., the quantified variables are polynomially bounded in length, and P is polynomial-time computable, construct the formula $\Phi'(B, \bar{d})$,

$$(\exists B', h)(B' \leqslant B \wedge B \leftarrow^h\!\to B' \wedge (\forall B'', k)(B \leftarrow^k\!\to B'' \to B' \leqslant B'') \wedge \Phi(B', h(\bar{x})))$$

(this is essentially the same construction taking M to M' above). If $i \geqslant 2$, Φ' can be converted to prenex form $(\exists y_1')(\forall y_2') \cdots (\Theta y_i')\ P'(B, \bar{x}, y_1', \ldots, y_i')$ and $\{(B, \bar{x}) \mid \Phi(B, \bar{x})\} \in \Sigma_i^P$. A formula $\Psi(B, \bar{x})$ is defined to be *consistent* if whenever $B \leftarrow^h\!\to B'$, $\Psi(B, \bar{x})$ iff $\Psi(B', h(\bar{x}))$. Then it can be seen that Φ' is consistent, and that whenever Φ is consistent, then $\Phi(B, \bar{x})$ iff $\Phi'(B, \bar{x})$. Thus the Φ' formulas provide an effective enumeration for $Q\Sigma_i^P$ for $i \geqslant 2$. The case for $Q\Pi_i^P$ is similar, only this time Φ' is constructed as

$$(\forall B', h)((B' \leqslant B \wedge B \leftarrow^h\!\to B' \wedge (\forall B'', k)(B \leftarrow^k\!\to B'' \to B' \leqslant B'')) \to \Phi(B', h(\bar{x}))),$$

and the case of PQHIER follows from either of these constructions. ∎

The next theorem states that the complexity classes for queries are ordered in much the same way as the standard complexity classes.

DEFINITION. We say that a class C is *closed under logspace reducibility* if whenever S_1 is in C and S_2 is many-one logspace reducible [19, 21] to S_1, then S_2 is in C, and $C \neq \{\{\ \}, \Sigma^*\}$ for alphabet Σ. All the usual complexity classes (such as LOGSPACE, PTIME, PHIER, PSPACE, etc.) are closed under logspace reducibility.

THEOREM 5.3. *If* C_1, C_2 *are closed under logspace reducibility, then* $QC_1 \subset QC_2$ *iff* $C_1 \subset C_2$.

Proof. The if-part is immediate. For the other direction, assume $C_1 \not\subset C_2$. Then there is a set $S \subset \{0,1\}^*$ in C_1 but not in C_2. For any $x \in \{0,1\}^*$, $x = x_1 x_2 \cdots x_k$, let B_x be a data base of type $(2,1)$ having domain $D = \{a_1, \ldots, a_k\}$ and $R_1 = \{(a_i, a_{i+1}) \mid 1 \leqslant i < k\}$ and $R_2 = \{a_i \mid x_i = 1\}$. From logspace reducibility, $S \in C_1$ (resp. C_2) iff $\{(B, (\,)) \mid \exists x \in S, h$ such that $B \leftarrow^h \rightarrow B_x\} \in C_1$ (resp. C_2). This follows by observing that an x such that $B \leftarrow^h \rightarrow B_x$ can be found in log space. Let Q be the query of type $(2,1) \rightarrow 0$ such that $Q(B) = \{(\,)\}$ if $B \leftarrow^h \rightarrow B_x$ for some h and some $x \in S$, and $Q(B) = \{\,\}$ otherwise. Then $Q \in QC_1$, but not $Q \in QC_2$. ∎

We now compare the complexity classes of queries with FP and SO (the set of second-order definable queries, Section 4).

THEOREM 5.4. FP $\subset\neq$ QPTIME $\subset$ QPHIER = SO $\subset\neq$ CQ.

Remark. The question "QPTIME = QPHIER?" is open. From Theorem 5.3, the answer is "yes" iff PTIME = PHIER which in turn holds iff PTIME = NPTIME (or, in usual terminology, $P = NP$) which is an outstanding open problem. Also, QPHIER $\subset$ QPSPACE, but QPHIER = QPSPACE iff PHIER = PSPACE, which is also an open problem of complexity theory.

Proof. (i) FP $\subset$ QPTIME. We show by induction on the structure of formulae Φ in LY (the language of fixpoint formulae) that

. For every appropriate $\bar{x}$, $\bar{R}$, the query represented by $\bar{x}.\bar{R}.\Phi$ is in QPTIME. $(*)$

Property $(*)$ holds for the atomic formulae, and if it holds for Φ, Ψ, it is immediately seen to hold for $\Phi \vee \Psi$, $\neg\Phi$, and $(\exists x)\Phi$ (because the number of possible values for x is no more than the size of the data base), and hence for $\Phi \wedge \Psi$, $(\forall x)\Phi$. Finally, if $(*)$ holds for $\Phi(R, \bar{x}, \bar{y})$, we show that it also holds for $(\bar{z}.YR(\bar{x}))\,\Phi(R, \bar{x}, \bar{y})$. Denote the latter formula by Ψ. Given any assignment θ mapping the free variables of Ψ into domain elements, the construction $\bigcup_i T_i$ of Theorem 4.1 can be done in polynomial time. Let n be the number of elements in the domain of the data base and R have rank a, then given any assignment θ' to variables in $\bar{x}$, $\bar{y}$, and relation T for R, if it can be determined whether or not $\Phi(T, \theta'(\bar{x}, \bar{y}))$ holds in time $P(\|(B, T)\|)$ for some polynomial P, then given T_i, T_{i+1} can be computed in time $n^a P(\|(B, T_i)\|)$, and since tuples are never deleted from T_i, the process stops at $T = T_{n^a}$; hence checking whether or not $\Psi(\theta(\bar{z}, \bar{y}))$ holds in B takes time polynomial in $\|B\|$, thereby proving that property $(*)$ holds for Ψ.

(ii) FP $\neq$ QPTIME. This is shown in the next section by showing that for the set RQ of ranked queries, FP $\subset$ RQ, but QPTIME $\not\subset$ RQ.

(iii) QPTIME $\subset$ QPHIER is immediate from Theorem 5.3.

(iv) SO $\subset$ QPHIER. This is immediate upon observing that second-order quantifiers can be thought of as polynomially bounded quantifiers.

(v) QPHIER $\subset$ SO. See [21, pp. 7–8].

(vi) SO $\subset\neq$ CQ follows from (iv), (v), and Theorem 5.3. ∎

6. An Application

In this section we use the classes of queries of Sections 4, 5 to characterize a class of queries defined in quite a different manner. Other such characterizations can be fund in [5]. Chandra and Harel [6] demonstrated a query language QL which could express all the computable queries. The variables of the language take relations as values, and relational terms can be built up using relational operators. Programs in QL consist of assignment statements, composed by sequencing and a looping construct. A key aspect of QL which gives it its power is that variables are not typed; they can take relations of arbitrary rank as values. The *width* of a relation can then be used in unconventional ways, e.g., as a counter to simulate a Turing machine computation. A natural question that arises is *what happens if all variables are ranked*? We shall examine this question next.

Let the query language RQL be defined to contain the following:

Variables: $X_0^0, X_1^0, X_2^0, \ldots, X_0^1, X_1^1, \ldots, X_0^2, \ldots$.
Terms: E, R_i, X_i^a,
and if t, t_1, t_2 are terms, then so are the following:
$t_1 \times t_2$,
$t_1 \cup t_2$,
$\neg t$
$\mathrm{Proj}_i(t)$ for integer $i \geqslant 1$
$\mathrm{Perm}_\theta(t)$ for a permutation θ
Programs: $X_i^a \leftarrow t$, where t is a term
and if P, P_1, P_2 are programs, then so are the following:
(P_1, P_2),
while $X_i^a \neq \{\ \}$ *do* P.

Variable X_i^a has rank a. This superscript will be omitted when no confusion results. All variables are initialized to $\{\ \}$. All terms are also ranked. Here E has rank 2, and its value is $\{(d, d) \mid d \in D\}$, where $B = (D, \bar{R})$ is the given input data base. The term $t_1 \times t_2$ has value $\{(\bar{d}, \bar{e}) \mid \bar{d} \in t_1,\ \bar{e} \in t_2\}$; $t_1 \cup t_2$ is set union if t_1, t_2 have the same rank, and is empty otherwise; $\neg t$ has value $D^a - t$, where t has rank a; $\mathrm{Proj}_i(t)$ has value $\{(d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_a) \mid (d_1, \ldots, d_a) \in t\}$; and, given a permutation θ on $\{1, \ldots, a\}$, where t has rank a, the value $\mathrm{Perm}_\theta(t) = \{\theta(\bar{d}) \mid \bar{d} \in t\}$. The semantics of programs is straightforward (if t is not of rank a, then $X_i^a \leftarrow t$ assigns $\{\ \}$ to X_i^a). It may be noted that an *if-then-else* construct can be simulated in RQL, and will be used in constructions below, as will other constructs which are easily simulated (see [6]). Given a program, we shall associate with it an *output variable* X whose value will contain the output in case the computation terminates.

Definition. Given a program P, let Q_P denote the corresponding query, and let RQ denote the set $\{Q_P \mid P$ halts for all appropriate $B\}$.

It may be noted that the restriction to total queries is not a serious one. For one thing, these are generally the queries of interest; for another, programs in RQL cycle only by repeating values of all their variables after an a priori time bound, and a clock can be attached to stop the computation if it runs too long. Totality is also needed to be able to compare the queries with queries defined from complexity classes.

THEOREM 6.1. $\mathrm{FP} \subset \mathrm{RQ} \subset\neq \mathrm{QPSPACE}$.

Proof. $\mathrm{RQ} \subset \mathrm{QPSPACE}$ is immediate upon noting that the space needed for a typed relation is bounded by a polynomial, and $\mathrm{RQ} \neq \mathrm{QPSPACE}$ follows from Theorem 6.2 which is proved independently. We now prove that $\mathrm{FP} \subset \mathrm{RQ}$. The proof is by induction on the formulae in LY. It is easy to see how atomic formulae, connectives $\vee$, $\neg$, and quantifier $\exists$ (and hence also $\wedge$, $\forall$) can be represented simply by terms in RQL. In general, a formula $\Psi \in LY$ will be represented by a program $P \in \mathrm{RQL}$ with one *free* variable for each free relation symbol R in Ψ. Let Ψ be

$$(\bar{z}.YR(\bar{x}))\, \Phi(R, \bar{x}, \bar{y})$$

and assume for simplicity that all variables in $\bar{z}$ are distinct, and none is in $\bar{y}$.

Suppose $\Phi(X_1^a, \bar{x}, \bar{y})$ can be represented by a program $P(X^a)$ with output variable X_1^{a+b} ($|\bar{x}| = a$, $|\bar{y}| = b$), that $\Phi(X^a, \bar{d}, \bar{e})$ holds iff $(\bar{d}, \bar{e}) \in X_1^{a+b}$ when $P(X^a)$ terminates, provided its free variables are initialized appropriately. Program P can be modified to a program $P'(X^{a+b})$ with output variable X_1^{a+2b} having the following property: if X^{a+b} is initialized to

$$X^{a+b} = \bigcup_{v \in D^b} X_v^a \times \{v\},$$

where for each v, $X_v^a \subset D^a$, and if X_{1v}^{a+b} is the value of X_1^{a+b} upon completing execution of P with X^a initialized to X_v^a, then the value of X_1^{a+2b} upon termination of P' is

$$X_1^{a+2b} = \bigcup_{v \in D^b} X_{1v}^{a+b} \times \{v\}.$$

Essentially P' executes several computations of P in parallel, one for each $v \in D^b$. Let $P''(X^{a+b})$ with output variable X_2^{a+b} be the same as $P'(X^{a+b})$ except that upon termination

$$X_2^{a+b} = \{uv \mid v \in D^a,\ v \in D^b,\ uvv \in X_1^{a+2b}\}.$$

Then $\Psi(\bar{z}, \bar{y})$ can be represented (by Theorem 4.1, with simultaneous execution of T_i's there for each possible value of $\bar{y}$) by the program

$X^{a+b} \leftarrow \{\ \}$;
BIT $\leftarrow \{(\ )\}$;
While BIT $\neq \{\ \}$ *do*
$(P''(X^{a+b})$;
If $X_2^{a+b} \neq X^{a+b}$,
then $X^{a+b} \leftarrow X_2^{a+b}$
else BIT $\leftarrow \{\ \}$).

The output variable of the program is X_2^{a+b}. In case $\bar{z}$ in Ψ has repetitions or variables common with $\bar{y}$, that is handled by a post-processing step, restricting X_2^{a+b} to have appropriate columns be equal, and then projecting out the redundant columns. ∎

THEOREM 6.2. QPTIME $\not\subset$ RQ.

Proof. Consider the query EVEN which merely checks to see if the domain of the data base is even. Its type is $(\ ) \to 0$, i.e. $B = (D)$, and

$$\begin{aligned} \text{EVEN}(B) &= \{(\ )\}, && \text{if } |D| \text{ is even,} \\ &= \{\ \}, && \text{otherwise.} \end{aligned}$$

It is immediate that EVEN $\in$ QPTIME. We show that EVEN is not in RQ. Suppose there is a program P in RQL whose corresponding query $Q_P = \text{EVEN}$. Without loss of generality we may let terms in P have at most one operator. Let the maximum rank of any variable in P be a. Let B, B' have domains of size $a+1$, $a+2$, respectively. We show that the computation paths of P on B and on B' are identical and so are the outputs (which is the desired contradiction). Given an equivalence relation EQV on $\{1, 2, \ldots, b\}$, let the query Q_{EQV} be represented by

$$(x_1, \ldots, x_b).(\ ).\ \bigwedge x_i = x_j \text{ (for } (i,j) \in \text{EQV}) \wedge \bigwedge x_i \neq x_j \text{ (for } (i,j) \notin \text{EQV}).$$

Note that if $\text{EQV}_1 \neq \text{EQV}_2$, then $Q_{\text{EQV}_1}(B) \cap Q_{\text{EQV}_2}(B) = \{\ \}$, and

$$\bigcup_{\text{EQV}} Q_{\text{EQV}}(B) = D_b.$$

Given a set S of equivalence relations on $\{1, \ldots, b\}$, let Q_S denote the query

$$Q_S(B) = \bigcup_{\text{EQV} \in S} Q_{\text{EQV}}(B).$$

Induction is on i, the number of steps of the computation. The inductive hypothesis is that i steps of the computation paths of P on B and B' are identical (a step is an assignment statement or a check $X \neq \{\ \}$ in a *while-do*), and after i steps, for each variable X in P there is a set S of equivalence relations on $\{1, \ldots, b\}$ (where b is the rank of X) such that the value of X in the computation on B (resp. B') is Q_S (B)

(resp. $Q_S(B')$). This is trivially true for $i = 0$. Now suppose it holds for i steps. If the $(i+1)$th step is a check $X \neq \{\ \}$, then it is true for $i+1$ steps since for every EQV on $\{1,...,b\}$, $b \leqslant a$, $Q_{\text{EQV}}(B) \neq \{\ \}$ and $Q_{\text{EQV}}(B') \neq \{\ \}$. If the $(i+1)$th step is an assignment statement whose term has no operator or has any of the five operators, the induction hypothesis can be seen to hold after $i+1$ steps. The result follows since when B, B' halt, if X (of rank 0) is the output variable of P, the value of X in B is $\{\ \}$ iff the value of X in B is also $\{\ \}$. ∎

THEOREM 6.3. *There is a* $Q \in RQ$ *such that* $\{B, \bar{x} \mid \bar{x} \in Q(B)\}$ *is logspace complete in* PSPACE.

Proof. We show how a program in RQL can simulate any linear space bounded Turing machine. The simulation of a machine computing a language complete in PSPACE will then produce the desired query.

Assume we are given a one-tape Turing machine with tape symbols $\Sigma = \{\sigma_0, \sigma_1,..., \sigma_p\}$, states $S = \{s_0,..., s_q\}$ and starting state s_0, accepting state s_q and rejecting state s_{q-1}, which never attempts to read a tape square beyond the squares on which the self delimiting input of length n is written, and which always halts and does so in state s_q or s_{q-1}. We describe the desired program below, after showing the correspondence between inputs and data bases. For input $x = x_0, x_1,..., x_{n-1}$, $x_i \in \Sigma$ (the x_i's may be further restricted if desired), the corresponding data base consists of the following relations (the domain contains exactly all the elements appearing in the relations):

for each $\sigma \in \Sigma$, a unary relation $\sigma = \{\sigma'\}$,
for each $s \in S$, a unary relation $s = \{s'\}$,
a binary relation
$\quad \text{Suc} = \{(x'_0, x'_1), (x'_1, x'_2),..., (x'_{n-2}, x'_{n-1})\}$,
a unary relation $\text{Head} = \{x'_0\}$
and a binary relation
$\quad \text{Tape} = \{(x'_i, \sigma'_j) \mid x_i = \sigma_j\}$.

The program maintains variables TAPE, HEAD, and STATE whose values correspond respectively to the tape contents, head position, and the Turing machine state during the computation. Recalling that D is the domain and can be obtained by $\text{Proj}_1 E$, and that Proj_i deletes the ith column, the program is:

TAPE ← Tape;
HEAD ← Head;
STATE ← s_0;
While STATE − $(s_q \cup s_{q-1}) \neq \{\ \}$ *do*
(SYMBOL ← Proj_1(HEAD × $D \cap$ TAPE);
If STATE = $s_0 \wedge$ SYMBOL = σ_0, *then* Action(s_0, σ_0)
else if STATE = $s_0 \wedge$ SYMBOL = σ_1 *then* Action(s_0, σ_1)
⋮
else if STATE = $s_i \wedge$ SYMBOL = σ_j, *then* Action(s_i, σ_j)...);
OUTPUT ← Proj_1(STATE $\cap$ s_q).

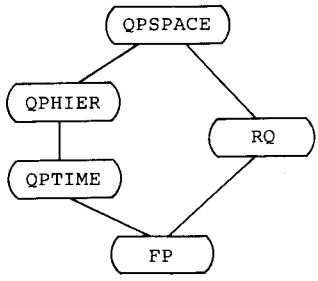

FIG. 2. The Hasse diagram relating RQ with other query classes (under the set inclusion ordering) upon the assumption PTIME $\neq$ PHIER $\neq$ PSPACE.

Action(s_i, σ_j) is as follows. In state s_i with the Turing machine head on symbol σ_j, the Turing machine may do one of the following:

(i) Write symbol σ and change state to s. Then Action is

$$(\text{TAPE} \leftarrow (\text{TAPE} - \text{HEAD} \times \text{SYMBOL}) \cup \text{HEAD} \times \sigma;$$
$$\text{STATE} \leftarrow s)$$

(ii) More the head left and change state to s. Then Action is

$$(\text{HEAD} \leftarrow \text{Proj}_2(D \times \text{HEAD} \cap \text{Suc});$$
$$\text{STATE} \leftarrow s)$$

(iii) Move the head right and change state to s. Then Action is

$$(\text{HEAD} \leftarrow \text{Proj}_1(\text{HEAD} \times D \cap \text{Suc});$$
$$\text{STATE} \leftarrow s).$$

It is easily seen by induction that TAPE, HEAD, and STATE simulate the Turing machine computation, that the program always halts, and that the output variable OUTPUT has value $\{\ \}$ iff the input is accepted by the Turing machine. ∎

A consequence of Theorem 6.3 (along with Theorem 6.1) is

COROLLARY 6.4. (i) RQ $\subset$ QPTIME *iff* PTIME = PSPACE,

(ii) RQ $\subset$ QPHIER *iff* PHIER = PSPACE.

It is generally conjectured that PTIME $\neq$ PHIER $\neq$ PSPACE. Under this assumption, RQ is independent (in a set containment ordering) of QPTIME, QPHIER, see Fig. 2.

7. CONCLUSIONS

The purpose of this paper has been to provide a framework for comparing and classifying sets of queries and query languages. As such, it is shown that the conjunctive [9] and tableau queries [1], the first-order queries and the relational

algebra [4], a query language suggested in [2], and the closure of first-order queries under composition and least fixpoint (i.e., FP) fall naturally onto an ω^2 hierarchy. The first ω levels of the hierarchy characterize the first-order queries, and are shown to be strict. All the queries in FP can be computed in polynomial time, but there are simple queries computable in polynomial time which are not in FP. In particular, it can be shown that queries in FP cannot count the number of tuples in a relation. Formally it is shown that no query in FP can determine whether the domain of the data base has an even number of elements. The concept, however, is quite general. It can be shown, e.g., that given a relation R (student-name, course) which gives the names of students in various courses, the following query is not in FP: *are there two courses with the same number of students*? Neither is the query *what is the course with the maximum attendance*? In fact, this limitation is not just of FP, but applies generally to query languages based on variables of bounded rank. All these queries can, of course, be computed in polynomial time. Unfortunately, we do not know of an effective enumeration of the queries computable in polynomial time, although such enumerations do exist for queries defined on larger complexity classes such as the polynomial-time hierarchy, polynomial space, etc. In fact, queries defined from the polynomial-time hierarchy turn out to be precisely the second-order queries. These classes of queries can be used to characterize the set of queries RQ computable using only ranked variables. The set of queries RQ turns out to be between FP and the set of queries computable in polynomial space. It is an open problem as to whether FP $\neq$ RQ (though this is implied by PTIME $\neq$ PSPACE).

Several interesting problems remain to be solved. These include showing the strictness of the j-fold fixpoint hierarchies, the strictness between these hierarchies for increasing j's, and obtaining an enumeration of the polynomial time queries (QPTIME).

Of more pragmatic interest is the problem of characterizing the expressive power of various constructs used in query languages. As we have observed, fixpoints do not provide the ability to count the size of a relation, and it is, therefore, worth exploring the limits of their usefulness. The use of ranked variables is another construct, which has only been explored in a preliminary manner in this paper. Other primitives suitable for study include the use of the equality relation, counters, and looping constructs in general. Results in this direction appear in [5].

It is possible that some of the classes of queries in this paper could provide an appropriate foundation for useable query languages. As such, it is interesting to ask whether there is a *natural* query language that can express (exactly?) all the queries in FP (or in QPTIME, QPSPACE, or some other complexity class).

An area of deep significance is that of obtaining some understanding of the set of queries computable using resources less than polynomial time. Of particular interest would be the queries computable in log time since most practical queries on data bases use indices to search for some record, and thereby take time about the log of the size of the data base. The primary difficulty in this is the formalization of the appropriate notions in a robust enough manner.

APPENDIX

a, b, c	Rank of a relation.
A	Formula.
B	Data base.
$B \leftarrow^{h} \rightarrow B'$	h isomorphism mapping B to B'.
C	Complexity class.
CQ	Set of computable queries.
D	Domain of a data base.
E	Set of existential queries.
Exist	Set of existential expressions.
F	Set of first-order queries.
First	Set of first-order expressions.
Fixpoint	Set of fixpoint expressions.
FP	Set of fixpoint queries.
$FP^{[j]}$	Restricted fixpoint queries.
L	Language of first-order formula.
LY	Language of fixpoint formulae.
P	Set of projection queries.
PTIME, etc.	Languages recognized in polynomial time, etc.
Q	Query.
Q_A	Query represented by formula A.
QC	Set of queries in C.
QPTIME, etc.	Queries computable in polynomial time, etc.
R	Relation.
RQ	Set of ranked queries.
RQL	Ranked query language.
S	Set of relations.
SO	Set of second-order queries.
T	Relation.
TC	Transitive closure query.
$TC^{[j]}$	j-fold Transitive closure query.
U	Universal domain.
YA	Set of fixpoints of A.
YF	Fixpoints of first-order queries.
$Y\Gamma$	Set of fixpoints of expressions in Γ.
$\neg$	Complement of a query/set of queries. negation of a formula/set of formulae.
$\circ$	Composition of queries/sets of queries. substitution in a formula/set of formula.
Γ, Δ	Set of formulae.
Θ	Quantifier, either $\exists$ or $\forall$.
Σ_i, Π_i	Hierarchy of expressions.
Σ_i^P, Π_i^P	Polynomial-time hierarchy.

$\Sigma_\alpha^Q, \Pi_\alpha^Q$	First-order, fixpoint hierarchies of queries.
$\Sigma_\alpha^{Q[J]}, \Pi_\alpha^{Q[J]}$	Restricted Fixpoint hierarchies.
Φ, Ψ	Formula.
$\lvert \cdots \rvert$	Number of elements in vector ..., e.g., $\lvert \bar{x} \rvert$.
$\lVert \cdots \rVert$	Length of the encoding of ..., e.g., $\lVert (B, \bar{x}) \rVert$.

ACKNOWLEDGMENT

We appreciate several helpful comments made by Moshe Vardi.

REFERENCES

1. A. V. AHO, Y. SAGIV, AND J. D. ULLMAN, Equivalences among relational expressions. *SIAM J. Comput.* **8** (1979), 218–246.
2. A. V. AHO AND J. D. ULLMAN, Universality of data retrieval languages, *in* "Proceedings, 6th ACM Symp. on Principles of Programming Languages," San Antonio, Texas, Jan. 1979, pp. 110–117.
3. E. F. CODD, A relational model of data for large shared data banks, *Comm. ACM* **13** (6) (1970), 377–387.
4. E. F. CODD, Relational completeness of data base sublanguages, *in* "Data Base Systems" (Rustin, Ed.), Prentice–Hall, Englewood Cliffs, N.J., 1972.
5. A. K. CHANDRA, Programming primitives for database languages, *in* "Proc. 8th ACM Symp. on POPL," Williamsburg, Virginia, Jan. 1981, pp. 50–62.
6. A. K. CHANDRA AND D. HAREL, Computable queries for relational data bases, *J. Comp. System Sci.* **21** (1980), 156–178.
7. A. K. CHANDRA AND D. HAREL, Structure and complexity of relational queries, *in* "Proc. 21st FOCS," Syracuse, New York, Oct. 1980, pp. 333–347.
8. A. K. CHANDRA AND D. HAREL, Horn clauses and the fixpoint query hierarchy, *in* "Proc. ACM Symp. on Principles of Database Systems," March 1982.
9. A. K. CHANDRA AND P. M. MERLIN, Optimal Implementation of Conjunctive Queries in Relational Data Bases *in* "Proceedings, 9th ACM Symp. on Theory of Computing," Boulder, Colorado, May 1977.
10. A. EHRENFEUCHT, An application of games to the completeness problem for formalized theores, *Fund. Math* **49** (1961), 129–141.
11. R. FAGIN, Generalized first-order spectra and polynomial-time recognizable sets, *Proc. SIAM–AMS* **7** (1974), 43–73.
12. R. FAGIN, Monadic generalized spectra, *Z. Math. Logic Grundlag. Math.* **21** (1975), 89–96.
13. R. FRAÏSSÉ, Sur les classifications des systèmes de relations, Publications Sc. d l'Université D'Alger 1, no. I, 1954.
14. H. GAIFMAN, Private Communication.
15. H. GALLAIRE AND J. MINKER (Eds.), "Logic and Data Bases," Plenum, N.Y., 1978.
16. N. IMMERMAN, Relational queries computable in polynomial time, *in* "Proc., ACM Symp. on Theory of Computing," May 1982.
17. R. A. KOWALSKI, Predicate logic as a programming language, *in* "Proc., IFIP, pp. 556–574, North-Holland, Amsterdam, 1974."
18. H. J. KEISLER AND W. WALKOE, JR., The diversity of quantifier prefixes. *J. Symbolic Logic* **38** (1) (1973), 79–85.
19. J. C. LIND, "Computing in Logarithmic Space," Tech. Memo. 52, M.I.T., Project MAC, Cambridge, Mass, 1974.

20. H. ROGERS, "Theory of Recursive Functions and Effective Computability," McGraw–Hill, N.Y., 1967.
21. L. J. STOCKMEYER, The polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977), 1–22.
22. M. H. VAN EMDEN, Computation and deductive information retrieval, *in* "Formal Description of Programming Concepts" (E. Neuhold, Ed.), pp. 421–439, North-Holland, Amsterdam, 1978.
23. M. VARDI, The complexity of relational query languages, *in* "Proceedings, 14th ACM Symp. on Theory of Computing," San Francisco, May 1982, pp. 137–146.
24. M. ZLOOF, Query by example, RC4917, IBM Research, Yorktown Heights, July 1974.
25. M. ZLOOF, Query by example, Operations on the transitive closure. RC5526, IBM Research Yorktown Heights, Oct. 1976.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium