HORN CLAUSE QUERIES AND GENERALIZATIONS*

ASHOK K. CHANDRA** AND DAVID HAREL†

▷ A logic program consists of a set of Horn clauses, and can be used to express a query on relational data bases. It is shown that logic programs express precisely the queries in YE^+ (the set of queries representable by a fixpoint applied to a positive existential query). Queries expressible by logic programs are thus not first-order queries in general, nor are all the first-order queries expressible as logic programs. Several ways of adding negation to logic programs are examined. The most general case is where arbitrary first-order formulas (with "nonterminal" relation symbols) are allowed. The resulting class has the expressive power of universally quantified second-order logic. ◁

1. INTRODUCTION

Kowalski [12] has introduced a programming language based on predicate calculus. A computation in this language is analogous to a resolution-driven attempt at proving a theorem of the form "atomic formula C follows from sentences $C_1, \ldots, C_m$". In [12], as well as in subsequent papers on the topic, e.g., [1, 6, 7], the sentences C_i are taken to be (closures of) Horn clauses in the predicate calculus; i.e., each C_i is of the form

$$(\forall \bar{x})(A \vee \neg B_1 \vee \cdots \vee \neg B_n)$$

for some atomic formulas A, B_j, where $\bar{x}$ consists of all variables appearing in A and in the B_j. Such a sentence is usually written simply as

$$A \leftarrow B_1, \ldots, B_n,$$

*A preliminary version of this paper, "Horn Clauses and the Fixpoint Query Hierarchy," appeared in the *ACM Symp. on Principles of Database Systems*, 1982.

**IBM Thomas J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598.

†Research of this author supported in part by a Bath Sheva Fellowship.

Address correspondence to Prof. David Harel, Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel.

52 Vanderbilt Ave., New York, NY 10017 0743-1066/85/$03.30

omitting the quantifiers. This approach forms the basis for the programming language PROLOG [14, 15] (see also [4] for an extended bibliography).

In van Emden [6] and in many of the papers in [8], the programming language of Horn clauses is viewed as a language for defining and querying relational data bases. The predicate symbols in such a Horn program P are regarded as the names of relations in a database B. If $\underline{S}_0$ is some new designated predicate symbol, then the set of constant tuples $\bar{d}$ for which $\underline{S}_0(\bar{d})$ is provable from P is taken as the response of the query represented by P when applied to B.

In this paper the expressive power of the query language **H** of Horn programs is investigated (a brief justification for considering the query capabilities of Horn clauses, rather than their defining capabilities, can be found in Harel [9]). It is shown that the set of queries expressible in this language is properly contained in the set FP of Chandra and Harel [5] that consists of queries representable using first-order operators ($\exists, \vee, \neg$) and fixpoint operators. In fact, Horn queries are precisely YE^+, which is the set of queries representable by a fixpoint applied to a positive existential query [5]. Since FP is closed under complements but YE^+ is not (because its queries are all monotone), it follows that FP strictly contains **H**. Also, since **H** can express the transitive closure query that is not a first-order query [2], it follows that the expressive power of **H** is independent of the first-order queries.

Several generalizations are possible that allow greater expressive power. The most simple of these preserves the resolution-based motivation behind choosing just Horn clauses [12]. For example, we define **H'** to be Horn clause queries extended to allow negation applied only to the terminals; i.e., atomic formulas of the form $\neg R(t_1, \ldots, t_n)$ are allowed among the premises of clauses. The class **H'** is readily seen to express precisely the queries in YE, which is the set of queries representable by a fixpoint applied to any existential query. A more powerful generalization is obtained by allowing negated nonterminals in the premises as well. Care has to be exercised here in defining the semantics, however. For instance, what should be the meaning of the following program: $\{\underline{S}_0(x) \leftarrow \underline{R}(x); \underline{S}_0(x) \leftarrow \neg \underline{S}_0(x)\}$? This problem can be finessed by requiring that there be no "cycles" when negated nonterminals appear. We define a class **C** of *clausal programs* having this property. These programs turn out to express precisely the fixpoint queries FP.

Since a set of Horn clauses can be viewed simply as a conjunction of implications, i.e., as a particular kind of first-order formula with nonterminals, one may ask what happens if arbitrary first-order formulas with nonterminals are allowed as programs. This case is also studied using one particular semantics. The resulting queries turn out to be those expressed by universally quantified second-order logic.

2. HORN CLAUSE QUERIES

The following definition of the language **H** of Horn clause queries is a simplified version of that appearing in, e.g., [1, 7] tailored for uninterpreted relational data bases. First some definitions.

Let there be given a countable *universal domain* U. A *relational database* (*data base* for short) *of type* $\bar{a} = (a_1, \ldots, a_k)$, $k \geq 0$, $a_i \geq 0$, is a tuple $B = (D, R_1, \ldots, R_k)$ where D, the *domain* of B (sometimes denoted $D(B)$) is a finite nonempty subset of U, and for $1 \leq i \leq k$, $R_i \subset D^{a_i}$. A *query* Q of *type* $\bar{a} \to b$ is a partial function from

the set of databases of type $\bar{a}$ to subsets of U^b such that $Q(B) \subset (D(B))^b$ whenever it is defined. In other words, a query produces a finite relation on the domain of its argument. For this paper we take $b \geq 1$. Minor changes are needed to handle the case $b = 0$ as well.

In order to define the language **H**, we call elements of *U constants*, and assume we have an unlimited supply of *terminal relation symbols* $\underline{R}_0, \underline{R}_1, \ldots$, and *nonterminal relation symbols* $\underline{S}_0, \underline{S}_1, \ldots$ of various nonnegative arities, and *variables* $x, y, z, x_1, \ldots$. We assume that $=$ and $\neq$ are special binary terminal relation symbols. A *term* is either a variable or a constant. For an n-ary relation symbol $\underline{R}$ (resp. $\underline{S}$, $=$, $\neq$) and terms $t_1, \ldots, t_n$, $\underline{R}(t_1, \ldots, t_n)$ (resp. $\underline{S}(t_1, \ldots, t_n)$, $t_1 = t_2$, $t_1 \neq t_2$) is an *atomic formula*. An atomic formula of the form $\underline{S}(t_1, \ldots, t_n)$ is called a *nonterminal atomic formula*. Denote by Γ the set of all variable-free atomic formulas. A *clause* C is an expression of the form

$$A \leftarrow B_1, \ldots, B_n,$$

$n \geq 0$, where A, the *conclusion* of **C**, is a nonterminal atomic formula, and $B_1, \ldots, B_n$, its *premises*, are atomic formulas. A clause with conclusion A will also be called an *A-clause*. A *program* P of **H** is a finite nonempty set of clauses in which there are no occurrences of constants.

A *valuation* θ is a function from variables to constants, and if A is an atomic formula, $A\theta$ is the result of replacing in A each variable x by $\theta(x)$. A valuation θ is called *D-restricted* for $D \subset U$ if $\theta(x) \in D$ for every x. We will think of a valuation θ as being defined on constants too, with $\theta(d) = d$ for each $d \in U$.

The intuition behind a program P can be described as follows. P represents the conjunction of its clauses. Each clause $A \leftarrow B_1, \ldots, B_n$ is taken to stand for the universal closure of the implication $(B_1 \wedge B_2 \wedge \cdots \wedge B_n) \supset A$, and the set of tuples in a nonterminal relation S is taken to be those $\bar{d}$ appearing in any atomic formula of the form $\underline{S}(\bar{d})$ whose truth is a consequence of P.

More formally, let P be a program of **H** with terminal relation symbols from among $=, \neq, \underline{R}_1, \ldots, \underline{R}_k$, the latter of arity $a_1, \ldots, a_k$, respectively. Let $B = (D, R_1, \ldots, R_k)$ be a database of type $\bar{a} = (a_1, \ldots, a_k)$. Define a set $T_B \subset \Gamma$ by

$$T_B = \bigcup_{p=0}^{\infty} T_p,$$

where

$$T_0 = \left\{ \underline{R}_i(d_1, \ldots, d_{a_i}) \mid 1 \leq i \leq k;\ d_j \in D \quad \text{for } 1 \leq j \leq a_i;\ (d_1, \ldots, d_{a_i}) \in R_i \right\}$$
$$\cup \{ = (d, d) \mid d \in D \}$$
$$\cup \{ \neq (d, d') \mid d, d' \in D, d \neq d' \}$$

$$T_{p+1} = T_p \cup \{ A\theta \mid \theta \text{ is a } D\text{-restricted valuation such that } A \leftarrow B_1, \ldots, B_n \text{ is in } P \text{ and } B_i\theta \in T_p \text{ for } 1 \leq i \leq n \}.$$

It is obvious that T_B is finite since P is finite, and hence includes only a finite number of relation symbols, and the only atomic formulas considered in the T_p are those in which all terms are constants in the finite set D.

In order to view P as representing a query on relational databases, one needs to identify one of the nonterminal relation symbols of P as the *carrier* that produces

the result. The carrier will usually be denoted by $\underline{S}_0$. Thus, if $\underline{S}_0$ is of arity b we define Query_P, the *query* (of type $\bar{a} \to b$) *represented* by P, simply by

$$\text{Query}_P(B) = T_B/\underline{S}_0$$

where

$$T_B/\underline{S} = \{(d_1, \ldots, d_a) \mid \underline{S}(d_1, \ldots, d_a) \in T_B\}$$

for any nonterminal $\underline{S}$ of arity a. We also say that Query_P is *definable* by P.

Example 2.1. Let P_1 consist of the clause

$$\underline{S}_0(x, y) \leftarrow \underline{R}_1(x, z), \underline{R}_2(z, y),$$

and let $B_1 = (D, R_1, R_2)$ be a relational data base of type $(2, 2)$. By the construction above one can see that for $p > 1$, $T_p = T_1$, and $T_1 - T_0 = \{\underline{S}_0(a, b) \mid a, b \in D$ and there is $c \in D$ such that $(a, c) \in R_1$ and $(c, b) \in R_2\}$. In other words, P_1 represents the query of type $(2, 2) \to 2$ that produces the relational composition of R_1 and R_2, denoted $R_1 \circ R_2$.

Example 2.2. Let P_2 consist of the clauses

$$\underline{S}_0(x, y) \leftarrow x = y,$$

$$\underline{S}_0(x, y) \leftarrow \underline{R}_1(x, z), \underline{S}_0(z, y),$$

and let $B_2 = (D, R_1)$ be a database of type (2). One can show that $T_1 - T_0 = \{\underline{S}_0(a, a) \mid a \in D\}$ and $T_{p+1} - T_p = \{\underline{S}_0(a, b) \mid (a, b) \in R_1 \circ R_1 \circ \cdots \circ R_1$, p times, denoted $R_1^p\}$. Thus $\text{Query}_{P_2}(B) = TC(B) = R_1^*$, where R_1^* is the reflexive transitive closure of R_1. *Note:* the first clause of P_2 could equivalently be written simply as $\underline{S}_0(x, x) \leftarrow$, with the right-hand side being empty.

3. SIMPLIFYING THE FIXPOINT STRUCTURE OF H

The definition of Query_P in the previous section is closely related to the "operational" semantics of *SLD-resolution* [12]. Using the latter approach one concludes that $\underline{S}(\bar{d})$ follows from P if the assumption that $\underline{S}(\bar{d})$ is false (written as the *negative clause* $\leftarrow \underline{S}(\bar{d})$) is refutable from P using SLD-resolution. A step in the refutation procedure corresponds to adding an element to T_{p+1} (see [1, 7]).

The outcome of P_2 in Example 2 can be viewed differently as the least relation S satisfying the fixpoint equation $S = I \cup (R_1 \circ S)$, where $I = \{(d, d) \mid d \in D\}$. This *least fixpoint* approach has indeed been rigorously established as an equivalent definition of the semantics of Horn clause programs (see [1, 7]).

Specifically, a program P can be thought of as consisting of a series of mutual relational equations. The values of the nonterminals are then taken to be the appropriate parts of the least solution, i.e., the least fixpoints.

Example 3.1. To illustrate mutual definitions, consider P_3:

$$\underline{S}_0(x, y) \leftarrow x = y$$

$$\underline{S}_0(x, y) \leftarrow \underline{R}_1(x, z), \underline{S}_1(z, y)$$

$$\underline{S}_1(x, y) \leftarrow \underline{R}_2(x, z), \underline{S}_0(z, y).$$

Here one can show that $\text{Query}_{P_3}(D, R_1, R_2) = (R_1 \circ R_2)^*$.

Our goal is to show that programs in **H** represent precisely the set of queries YE^+ of [5], consisting essentially of a single relational fixpoint operator applied to a positive first-order existential query. The main observation needed to show this is the possibility of simulating a mutual fixpoint definition by a simple one. For any program P in **H** we show in this section how to construct an equivalent program P_1 having only one instrumental nonterminal $\underline{S}_1$, from which the carrier $\underline{S}_0$ of P_1 is obtained directly by a simple projection.

The idea is to make $\underline{S}_1$ "wide" enough to contain all of the information in the n nonterminals of P in addition to a "code", of width $O(\log n)$, which indicates for each tuple which one of the nonterminals in P it corresponds to. A careful choice of the code for $\underline{S}_0$ helps overcome a technical difficulty when the domain of the database has only one element.

We first describe a preliminary construction for the case $n = 2$. Let P involve only the two nonterminals $\underline{S}_0$ and $\underline{S}_1$, of arities b and b_1, respectively, and let $\underline{S}$ be a new relation symbol of arity $\max(b, b_1) + 2$. Without loss of generality we can assume that $b_1 \geq b$. Let $b_1 = b + q$. For an atomic formula A appearing in P, define A^+ as follows:

$$A^+ = \begin{cases} A & A \text{ is terminal} \quad \text{(case 1)} \\ \underline{S}(\bar{t}, \bar{x}, v, v') & A = \underline{S}_0(\bar{t}) \quad \text{(case 2)} \\ \underline{S}(\bar{u}, v, v') & A = \underline{S}_1(\bar{u}) \quad \text{(case 3)}, \end{cases}$$

where $|\bar{x}| = q$, and v, v' and the elements of $\bar{x}$ are new variables. Now, construct P^+ by replacing each clause C of the form $A \leftarrow B_1, \ldots, B_m$ in P by C^+, constructed as follows. First, form $A^+ \leftarrow B_1^+, \ldots, B_m^+$. Then for each atomic formula $A, B_1, \ldots, B_m$, if the transformation to the + version was by case 2 above (respectively, case 3) add the atomic formula $v = v'$ (respectively, $v \neq v'$) to the premises. The final clause obtained in this way is C^+.

Example 3.2. Consider P_3 of Example 3.1, the construction yields P_3^+:

$$\underline{S}(x, y, v_1, v_2) \leftarrow x = y, v_1 = v_2,$$

$$\underline{S}(x, y, v_3, v_4) \leftarrow \underline{R}_1(x, z), \underline{S}(z, y, v_5, v_6), v_3 = v_4, v_5 \neq v_6$$

$$\underline{S}(x, y, v_7, v_8) \leftarrow R_2(x, z), \underline{S}(z, y, v_9, v_{10}), v_7 \neq v_8, v_9 = v_{10}. \qquad \square$$

The following can be proved routinely by induction on p from the definitions of T_p, T_p^+ and P^+:

Lemma 3.1. Let P^+ be constructed from P as above, and let $B = (D, \bar{R})$, $|D| > 1$. Let $T_B = \bigcup_p T_p$ and $T_B^+ = \bigcup_p T_p^+$ be the sets corresponding to P and P^+, given B as input. Then for every $p \geq 0$ and for every $\bar{d} \in D^b$, $\bar{e} \in D^q$, and $g, f \in D$ where $g \neq f$,

$$\bar{d} \in T_p/\underline{S}_0 \quad \textit{iff} \quad (\bar{d}, \bar{e}, f, f) \in T_p^+/\underline{S}, \tag{1}$$

$$(\bar{d}, \bar{e}) \in T_p/\underline{S}_1 \quad \textit{iff} \quad (\bar{d}, \bar{e}, f, g) \in T_p^+/\underline{S}. \tag{2}$$

In other words, if $I = \{(d, d) | d \in D\}$ then

$$T_B^+/\underline{S} = (T_B/\underline{S}_0 \times D^q \times I) \cup (T_B/\underline{S}_1 \times (D^2 - I)).$$

Note that the restriction on the size of D in the lemma prevents $D^2 - I$ from being

empty, and hence makes possible the representation for $\underline{S}_1$. Now, given P, denote by P_0 the program

$$P^+ \cup \{\underline{S}_0(x_1, \ldots, x_b) \leftarrow \underline{S}(x_1, \ldots, x_b, x_1, \ldots, x_1)\}$$

with carrier S_0. Call a database with domain D *trivial* if $|D| = 1$. As a direct corollary of Lemma 3.1 we have:

Lemma 3.2. For every nontrivial database B, $Query_{P_0}(B) = Query_P(B)$.

The construction of P^+ from P can easily be extended to the general case of n nonterminals, by taking $\underline{S}$ to be of arity $m + [(\log n) + 1$ (logarithms are base 2), where m is the maximum of the arities of the nonterminals. The additional $r = [(\log n) + 1$ components of $\underline{S}$ are used to encode the numbers between 0 and $n - 1$ by asserting (and consequently adding to the premises) equalities and nonequalities between their values. With variables $v, v_0, v_1, \ldots, v_{r-2}$, the "all-equal" code $v = v_{r-2}, v = v_{r-3}, \ldots, v = v_0$, is reserved for the carrier S_0. In general, the code for S_1 consists of $r - 1$ atomic formulas $A_{r-2}, \ldots, A_0$; and if the binary representation of i is $j_{r-2}, \ldots, j_0$ then A_k is $v = v_k$ if $j_k = 0$, otherwise A_k is $v \neq v_k$. The final program P_0 which is equivalent to P over nontrivial data bases is obtained similarly by projection:

$$P_0 = P^+ \cup \{\underline{S}_0(\bar{x}) \leftarrow \underline{S}(\bar{x}, x_1, \ldots, x_1)\}.$$

We would now like to remove the restriction on the size of the domain, admitting trivial databases too. As an illustration of a typical difficulty, consider the following.

Example 3.3. Let P_4 be

$$\underline{S}_0(x, y) \leftarrow \underline{R}_2(x, z), \underline{S}_1(z, y).$$

$$\underline{S}_1(x, y) \leftarrow x = y$$

$$\underline{S}_1(x, y) \leftarrow \underline{R}_1(x, z), \underline{S}_0(z, y)$$

In constructing P_4^+ the first clause is transformed into

$$\underline{S}(x, y, v, v_0) \leftarrow \underline{R}_2(x, z), \underline{S}(z, y, v', v_0'), v = v_0, v' \neq v_0'.$$

Now, since the rightmost atomic formula is never satisfied in a trivial database, the value of $\underline{S}_0$ in $(P_4)_0$ (i.e., P_4^+ with the projection clause) will be ϕ, whereas it should be $\{(d, d)\}$ whenever $R_2 = \{(d, d)\}$.

A clause C is 1-*derived* from a program P in **H** if it is obtained by simultaneously replacing, in a clause C' of P, each nonterminal atomic formula $S(\bar{x})$ from among the premises by the premises of an S-clause C'' in $P, C'' \neq C'$, after appropriately renaming the variables of C'' to match the elements of $\bar{x}$ and to be otherwise distinct from those of C'. C is *n-derived* from P, $n > 1$, if it is 1-derived from $P \cup Z$, where Z is the set of all clauses m-derived from P, for all $m < n$. Identifying clauses which are the same up to consistent renaming of variables, we let

$$P^* = P \cup \{C | C \text{ is } n\text{-derived from } P \text{ and } n < |P|\}.$$

Clearly, by the convention just adopted, P^* is finite, hence a program of **H**.

Example 3.4. Consider P_4 of Example 3.3. P_4^* consists of the following clauses in addition to the ones in P_4:

$\underline{S}_1(x, y) \leftarrow \underline{R}_1(x, z), \underline{R}_2(z, u), \underline{S}_1(u, y)$

$\underline{S}_0(x, y) \leftarrow \underline{R}_2(x, z), z = y$

$\underline{S}_0(x, y) \leftarrow \underline{R}_2(x, z), \underline{R}_1(z, u), \underline{S}_0(u, y)$

$\underline{S}_1(x, y) \leftarrow \underline{R}_1(x, z), \underline{R}_2(z, u), u = y$

$\underline{S}_1(x, y) \leftarrow \underline{R}_1(x, z), \underline{R}_2(z, u), \underline{R}_1(u, w), \underline{S}_0(w, y)$

$\underline{S}_0(x, y) \leftarrow \underline{R}_2(x, z), \underline{R}_1(z, u), \underline{R}_2(u, w), \underline{S}_1(w, y)$

$\underline{S}_0(x, y) \leftarrow \underline{R}_2(x, z), \underline{R}_1(z, u), \underline{R}_2(u, w), w = y$

the first three of which are 1-derived and the rest 2-derived, from P_4.

Clearly, since all we have done is to add clauses that are consistent with P, we have:

Lemma 3.3. For every database B, $Query_{P^*}(B) = Query_P(B)$.

However, we can now prove

Theorem 3.4. Let $(P^*)^+$ be constructed from P^* as described above for P, and let $P_1 = (P^*)^+ \cup \{\underline{S}_0(\bar{x}) \leftarrow \underline{S}(\bar{x}, x_1, \ldots, x_1)\}$. Then for every database B, $Query_{P_1}(B) = Query_P(B)$.

PROOF. For nontrivial B the result follows from Lemmas 3.2 and 3.3. Also, since, for trivial B (as illustrated in Example 3.3), all that can go wrong in the construction of $(P^*)^+$ is the *loss* of information, we obtain $\text{Query}_{P_1}(B) \subset \text{Query}_P(B)$.

Let B be a trivial database with domain $\{d\}$. Each relation symbol can thus take on at most two values, ϕ and $\{\bar{d}\} = \{(d, \ldots, d)\}$. According to which of these is the case we shall say that the relation symbol is *off* or *on*, respectively. To prove that $\text{Query}_P(B) \subset \text{Query}_{P_1}(B)$, assume that $\text{Query}_P(B) = \{\bar{d}\}$, i.e., that at some point in the construction of the sets T_p for P, $\underline{S}_0$ becomes on. Let p be the smallest such index, i.e., there is an $\underline{S}$-clause C in P, free of $\neq$, all of whose premises are on in T_{p-1}, but $\underline{S}_0$ (which, as a consequence, is on in T_p) is on in no $T_{p'}$ for $p' < p$. However, this means that there is an $\underline{S}_0$-clause C' that is p-derived from C, which has no nonterminal premises, and in which all premises are on in B.

We now argue that $p < |P|$, for if $p \geq |P|$ then some clause in P is used twice in the substitution process in an essential way (i.e., contributing to the length of the derivation process). Since there are only two possible values for the nonterminal in the conclusion of such a clause, the subderivation involving each such double usage can be "folded" by performing the substitutions following the second immediately after the first. This change can have no effect on the final outcome of $\underline{S}_0$ but shortens the derivation, contradicting the minimality of p. □

4. $\mathbf{H = YE^+}$

We first summarize some concepts of first-order queries and fixpoints. We will use a slightly modified version of the notation from [5]. The reader is urged to consult [5] for precise definitions.

From any first-order formula $\Phi(\bar{x})$ over relation symbols $=, \neq, \underline{R}_0, \underline{R}_1, \ldots$ and with, say, k free variables $\bar{x}$, one obtains a *first-order query* Q_Φ whose value on a data base B (of the right type) is

$$Q_\Phi(B) = \{\bar{d} \in (D(B))^k \mid \Phi(\bar{d}) \text{ is true in } B\}.$$

The set of first-order queries is denoted by F. Let E denote the set of *existential queries* representable by first-order formulas of the form

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

where Φ is quantifier-free. Let E^+ be the set of *positive existential queries* defined as E, except that Φ contains no negations. For example, the formula $(\exists z)(\underline{R}_1(x, z) \wedge \underline{R}_2(z, y))$ in E^+ represents $R_1 \circ R_2$ (compare with Example 2.1).

If Φ is a first-order formula involving relation symbols from among $\underline{S}, \underline{R}_0, \underline{R}_1, \ldots, \underline{R}_k$ then the formula

$$(\bar{z}.Y\underline{S})\Phi$$

(where all occurrences of S in Φ are positive, i.e., under an even number of negations) represents the query obtained by first taking the least fixpoint of the equation $\underline{S} = \Phi(\underline{S})$ and then matching the elements of $\bar{z}$ with the components of the resulting relation. Thus $\bar{z}$ can be used to force equality between some of the components of the fixpoint. Here the arity of $\underline{S}$, the length of $\bar{z}$, and the number of free variables in Φ are equal.[1] Hence, if the relation symbols free in such a formula Ψ ($\underline{S}$ is bound in $(\bar{z}.Y\underline{S})\Phi$) are $\underline{R}_1, \ldots, \underline{R}_k$ of arities $\bar{a} = (a_1, \ldots, a_k)$, and Ψ has b free variables (the number of distinct variables in $\bar{z}$), then Ψ represents a query of type $\bar{a} \to b$. YE^+ (resp. YE, YF) is the set of queries of this form where Φ is in E^+ (resp. E, F), i.e., representable by one application of the Y operator to a positive existential formula (resp. existential formula, any formula) of first-order logic.

Example 4.1. The reflexive transitive closure query TC is in YE^+ since it is represented by

$$\Psi_1\colon ((x, y).Y\underline{S})(\exists z)(x = y \vee (\underline{R}_1(x, z) \wedge \underline{S}(z, y))).$$

The similarity of this formula for TC and the program P_2 of Example 2.2 is the core of our main result:

Theorem 4.1. A query is definable by a program of **H** *iff it is in* YE^+.

PROOF. *If-part.* For a query Q in YE^+ represented by a formula Φ, define the program P_Φ in **H** inductively as follows: if Φ is $x = y$ then P_Φ is $\{\underline{S}_0(x, y) \leftarrow x = y\}$. Similarly for $\neq$ and $R_i(\bar{x})$. Let $\Phi(\bar{x}, \bar{y}, \bar{z}) = \Phi_1(\bar{x}, \bar{y}) \wedge \Phi_2(\bar{x}, \bar{z})$. For $i = 1, 2$ let P_i be P_{Φ_i} with the new relation symbol $\underline{S}_i$ replacing $\underline{S}_0$. Then

$$P_\Phi = P_1 \cup P_2 \cup \{\underline{S}_0(\bar{x}, \bar{y}, \bar{z}) \leftarrow \underline{S}_1(\bar{x}, \bar{y}), \underline{S}_2(\bar{x}, \bar{z})\}.$$

Similarly, if $\Phi(\bar{x}, \bar{y}, \bar{z}) = \Phi_1(\bar{x}, \bar{y}) \vee \Phi_2(\bar{x}, \bar{z})$, then

$$P_\Phi = P_1 \cup P_2 \cup \{\underline{S}_0(\bar{x}, \bar{y}, \bar{z}) \leftarrow \underline{S}_1(\bar{x}, \bar{y})\} \cup \{\underline{S}_0(\bar{x}, \bar{y}, \bar{z}) \leftarrow \underline{S}_2(\bar{x}, \bar{z})\}.$$

[1] It is necessary to show the equivalence of this version of the language and the slightly more elaborate one of [5], in which fixpoint formulas were of the form $(\bar{z}.Y\underline{S}(\bar{x}))\Phi(\bar{x}, \bar{y})$.

If $\Phi(\bar{x}) = \exists y \Phi_1(\bar{x}, y)$, then

$$P_\Phi = P_1 \cup \{\underline{S}_0(\bar{x}) \leftarrow \underline{S}_1(\bar{x}, y)\}.$$

Finally, if $\Phi(\bar{z}) = (\bar{z}.Y\underline{S})\Phi_1(\bar{x})$, then

$$P_\Phi = P_1 \cup \{\underline{S}(\bar{x}) \leftarrow \underline{S}_1(\bar{x})\} \cup \{\underline{S}_0(\bar{z}) \leftarrow \underline{S}(\bar{z})\}.$$

P_Φ can easily be seen to represent Q.

Only-if part. Given a program P in **H**, form the equivalent program $P_1 = (P^*)^+ \cup \{\underline{S}_0(\bar{x}) \leftarrow \underline{S}(\bar{x}, x_1, \ldots, x_1)\}$ of Section 3. Here $\bar{x} = (x_1, \ldots, x_m)$. The part of P_1 we are interested in is $(P^*)^+$ from which we now construct a formula Φ_P of the form $((\bar{x}, x_1, \ldots, x_1).Y\underline{S}_0)\Psi$, where Ψ is a positive existential formula. Observing that the free variables of Φ_P are only those of $\bar{x}$, and that the choice of the "all-equal" code for $\underline{S}_0$ in $(P^*)^+$ enables us to simulate the projection clause of P_1 in Φ_P, the latter will easily be seen to represent the same query as P_1, and hence by Theorem 3.4, the same query as P.

To construct Ψ notice that $\underline{S}$ is the only nonterminal in $(P^*)^+$, say its arity is r. First consistently change the variables of all the clauses of $(P^*)^+$ so that all conclusions are precisely $\underline{S}(\bar{y})$, where $\bar{y} = (y_1, \ldots, y_r)$ is a tuple of new variables. This can be done simply by replacing each clause $\underline{S}_0(z_1, \ldots, z_r) \leftarrow B_1, \ldots, B_n$ by the clause $\underline{S}_0(y_1, \ldots, y_r) \leftarrow B_1, \ldots, B_n, z_1 = y_1, \ldots, z_r = y_r$. Let the resulting set of clauses be $\{\underline{S}(\bar{y}) \leftarrow B_{i,1}, \ldots, B_{i,n_i}\}, 1 \le i \le p, 0 \le n_i$. Now define Ψ to be the formula

$$(\exists \bar{u})\Big(\bigvee_{1 \le i \le p} \bigwedge_{0 \le j \le n_i} B_{i,j}\Big),$$

where $\bar{u}$ consists of all variables in the above set of clauses except for those in $\bar{y}$. This construction is very similar to that given in [1] for constructing the "IF version" associated with a Horn-clause program. The reader should be able to see without difficulty that the fixpoint equation associated with Φ_P is essentially a conjunction of the meanings of the clauses of $(P^*)^+$, as described in the introduction. □

Example 4.2. The two directions in the proof above, if applied, respectively, to the representations P_2 and Ψ_1 of *TC* in Examples 2.2 and 4.1, yield (almost) one another. To be precise one obtains the following:

P_{Ψ_1}: $\underline{S}_0(x, y) \leftarrow \underline{S}(x, y)$
$\underline{S}(x, y) \leftarrow \underline{S}_1(x, y)$
$\underline{S}_1(x, y) \leftarrow \underline{S}_2(x, y, z)$
$\underline{S}_2(x, y, z) \leftarrow \underline{S}_3(x, y)$
$\underline{S}_2(x, y, z) \leftarrow \underline{S}_4(x, y, z)$
$\underline{S}_3(x, y) \leftarrow x = y,$
$\underline{S}_4(x, y, z) \leftarrow \underline{S}_5(x, z), \underline{S}(z, y)$
$\underline{S}_5(x, z) \leftarrow \underline{R}_1(x, z)$

Φ_{P_2}: $((x, y).Y\underline{S})((\exists z)(x = y \vee (\underline{R}_1(x, z) \wedge \underline{S}(z, y))))$.

The proof of Theorem 4.1 can be used to show that YE^+ is closed under its basic operations. This is done by translating from YE^+ to **H** using the "if" part of the

proof and then translating the new program (which involves the operation considered) back to YE^+ using the "only-if" part. One can prove this way that

Proposition 4.2. YE^+ *is closed under conjunction* ($\wedge$), *disjunction* ($\vee$), *composition* ($\circ$), *and fixpoint* (Y).

5. GENERALIZATIONS

The queries in YE^+ do not contain, and are clearly not closed under *negation* (or complementation, denoted $\neg$). In fact, the trivial query of type $(1) \to 1$ which complements its argument, yielding $D - R$ when applied to $B = (D, R)$, is not expressible in YE^+, and hence not in **H** either. Such minor "weaknesses" can be removed by defining the class **H'** of Horn clauses augmented with negated terminals.

A *literal* is an atomic formula ($\underline{R}(t_1, \ldots, t_n), \underline{S}(t_1, \ldots, t_n), t_1 = t_2, t_1 \neq t_2$), or has the form $\neg\underline{R}(t_1, \ldots, t_n)$. An *extended clause* C is an expression of the form $A \leftarrow B_1, \ldots, B_n, n \geq 0$, where A is a nonterminal atomic formula and the B_is are literals. A program P of **H'** is a finite nonempty set of extended clauses without constants. A program P represents a query Query_P as defined for **H** (Section 2) with the proviso for negations (by adding the set $\{\neg\underline{R}_i(d_1, \ldots, d_{a_i}) | (d_1, \ldots, d_{a_i}) \notin R_i\}$ to T_0).

Theorem 5.1. A query is definable by a program of **H'** *iff it is in* YE.

The proof is as that for Theorem 4.1. One needs to check that Lemma 3.3 and Theorem 3.4 also hold for programs in **H'**.

Even with this extension, **H'** cannot represent all first-order queries. The obvious feature missing is universal quantification. Let UNIV be the query of type $(1) \to 1$ whose value on $B = (D, R)$ is given by

$$\text{UNIV}(B) = \begin{cases} D & \text{if } R = D \\ \{\ \} & \text{otherwise.} \end{cases}$$

This is a first-order query: $\{x | \forall y \underline{R}(y)\}$ (for technical reasons we are not allowing queries of type $(1) \to 0$, otherwise we could have defined UNIV to have just a "yes/no" value).

Theorem 5.2. UNIV *is not in* YE.

This follows from the following property of queries in YE:

Lemma 5.3. Let $B = (D, R_1, \ldots, R_k)$, $B' = (D', R'_1, \ldots, R'_k)$ *be databases of the same type* $(a_1, \ldots, a_k)$, *with* $D \subset D'$, *and for all* i, $R_i = R'_i \cap D^{a_i}$. *Then if* Q *is in* YE, $Q(B) \subset Q(B')$.

PROOF. Let Q be represented by

$$((y_1, y_2, \ldots, y_n).Y\underline{S})\exists \bar{x}.\Phi(\underline{S}, \bar{x}, \bar{y}),$$

where the arity of $\underline{S}$ is n and $\bar{y} = (y_1, \ldots, y_n)$. Here Φ is quantifier free, and $\underline{S}$

appears only positively in Φ. The least fixpoint S satisfying $\underline{S}(\bar{y}) \equiv \exists \bar{x}.\Phi(\underline{S}, \bar{x}, \bar{y})$ on the database B is given by $S = \cup S_i$ where $S_0 = \{\ \}$, and $S_{i+1} = \{\bar{d} | \exists \bar{x}.\Phi(S_i, \bar{x}, \bar{d})$ is true in $B\}$. Let S', S_i' be the corresponding sets for B'. We show by induction on i that $S_i \subset S_i'$ (hence $S \subset S'$, from which the lemma follows). It is true for $i = 0$ where $S_0 = S_0' = \{\ \}$. Let $S_i \subset S_i'$, and suppose $\bar{d} \in S_{i+1}$. Hence for some vector $\bar{e}$ of elements in D, $\Phi(S_i, \bar{e}, \bar{d})$ is true in B. Since for all other relation symbols $\underline{R}_i$ appearing in Φ, relations R_i, R_i' are identical on the elements of $\bar{e}$, $\bar{d}$, and Φ is quantifier free, it follows (by an easy induction on the structure of Φ) that $\Phi(S_i, \bar{e}, \bar{d})$ is true in B'. Now, as $\underline{S}$ occurs only positively in Φ, Φ is monotone in the relation for $\underline{S}$, hence $\Phi(S_i', \bar{e}, \bar{d})$ is true in B', and $\bar{d} \in S_{i+1}'$. Thus $S_{i+1} \subset S_{i+1}'$, which completes the induction and the proof. □

PROOF OF THEOREM 5.2. Let $B = (D, R)$, $B' = (D', R')$, where $D = R = R' = \{d\}$, and $D' = \{d, d'\}$. Now UNIV$(B) = \{d\}$. If UNIV were in YE, by Lemma 5.3, $\{d\} \subset$ UNIV(B'), which is false since UNIV$(B') = \{\ \}$. □

There seem to be at least two ways to generalize **H'**. One is by allowing the negation of nonterminal symbols and introducing some notion of order on the "evaluation" of programs. The other is by attempting some semantics for general first-order formulas with nonterminal relation symbols considered as programs. We now consider the first possibility.

Let P_1 and P_2 be programs in **H'**, and let P_2' be P_2 with all nonterminals renamed to be distinct from those of P_1. In particular, assume that the n-ary carrier of P_2' is $\underline{S}_0'$. If $\underline{R}$ is an n-ary terminal of P_1 not appearing in P_2 (and hence not in P_2' either) we define the new program $P_1(\underline{R}/\neg P_2)$ to be $P_1 \cup P_2' \cup \{\underline{R}(\bar{x}) \leftarrow \neg \underline{S}'_0(\bar{x})\}$ for some n-tuple of variables $\bar{x}$. $\underline{R}$ now becomes a nonterminal in $P_1(\underline{R}/\neg P_2)$, and may be systematically renamed to some new S_i. The idea is simply to allow the complement of the **H'** query P_2 to be used in the **H'** query P_1. Let **C** be the language obtained from **H'** by inductively allowing new programs $P_1(\underline{R}/\neg P_2)$ for programs P_1 and P_2.

Since a program in **C** can be partially ordered into subprograms P_i such that $P_i < P_j$ iff P_j contains an occurrence of $\neg P_i$, the semantics of programs in **C** becomes straightforward: given $B = (D, \bar{R})$, evaluate the $P_i(B)$ in some order consistent with $<$, substituting in the process $D^n - P_j(B)$ for $\underline{R}$ whenever $P_i(\underline{R}/\neg P_j)$ occurs.

Next we need to define the general notion of fixpoint queries [5]. Special cases (YE$^+$, YE, YF) have already been defined. The class FP of *fixpoint queries* is defined using *fixpoint formulas*, which can be of one of the following forms: $\underline{R}(x_{i_1}, \ldots, x_{i_n})$, $\underline{S}(x_{i_1}, \ldots, x_{i_n})$, $\neg\Phi_1$, $\Phi_1 \vee \Phi_2$, $\Phi_1 \wedge \Phi_2$, $\exists x.\Phi_1$, $\forall x.\Phi_1$, $(\bar{z}.Y\underline{S})\Phi(\bar{x})$, where $\underline{R}$ is an n-ary terminal relation symbol (or is $=$, $\neq$), $\underline{S}$ is an n-ary nonterminal relation symbol, $x, x_{i_1}, \ldots, x_{i_n}$ are variables, Φ_1, Φ_2 are fixpoint formulas, $\Phi(\bar{x})$ is a fixpoint formula with n distinct free variables $\bar{x}$ and with $\underline{S}$ appearing only positively in it, and $\bar{z}$ is a vector of n (not necessarily distinct) variables. Variables of $\bar{z}$ are free in $(\bar{z}.Y\underline{S})\Phi(\bar{x})$, and $\bar{x}$, $\underline{S}$ are bound in it. A fixpoint formula $\Phi(\bar{x})$ with free relation symbols $R_0, R_1, \ldots$ represents a query Q_Φ in the obvious way:

$$Q_\Phi(B) = \{\bar{d} | \Phi(\bar{d}) \text{ is true in } B\}.$$

It remains only to explain the semantics of the least fixpoint operator Y. Given $(\bar{z}.Y\underline{S})\Phi(\bar{x})$, let $\bar{z}'$ be a vector of the distinct variables of $\bar{z}$; say $\bar{z}' = (z_1, \ldots, z_b)$, $\bar{z} =$

$(z_{i_1}, \ldots, z_{i_n})$. Also, for $\bar{d}' = (d_1, \ldots, d_b) \in D(B)^b$, let $\bar{d}$ be $(d_{i_1}, \ldots, d_{i_n})$. Now the formula $(\bar{z}.Y\underline{S})\Phi(\bar{x})$ is true for $\bar{d}'$ (as the value for $\bar{z}'$) iff $\bar{d} \in S_0$, where S_0 is the smallest relation $S \subset D(B)^n$ such that $\bar{e} \in S$ iff $\Phi(\bar{e})$ is true in B (with S being the relation for $\underline{S}$ in Φ).

Theorem 5.4. A query is definable by a program of **C** *iff it is in* FP.

The proof in one direction is by induction on the structure of programs (and using Theorem 5.1); in the other direction by induction on the structure of fixpoint formulas.

A result of Clark [3] (see also [1]) justifies the extension of **H** to **C** from an operational point of view. Clark shows that the complement of R consists of all tuples $\bar{d}$ such that an attempted proof of $\underline{R}(\bar{d})$ using SLD resolution fails on all branches of the proof tree in a finite amount of time. Hence, computing $\neg R$ by this "finite-failure" method is a natural extension of the resolution-based method for computing queries in **H**, giving rise to the possibility of using the language **C**.

The other possible generalization of **H'** mentioned earlier is to allow general first-order formulas as programs. A *first-order formula with nonterminals* $\Phi(\underline{S}_0, \underline{S}_1, \ldots)$ (without free variables) over relation symbols $\underline{S}_0, \underline{S}_1, \ldots, \underline{R}_0, \underline{R}_1 \ldots, =, \neq$, and *carrier* $\underline{S}_0$ represents a query Query_Φ given by

$$\text{Query}_\Phi(B) = \cap \{ S_0 | \exists S_1, \ldots, \text{s.t. } \Phi(S_0, S_1, \ldots) \text{ is true in } B \}.$$

Let FN denote the class of such queries.

It may be seen that this definition is consistent with those for **H**, **H'** (where a Horn clause is treated as a universally quantified implication, and a program is a conjunction of its clauses). It is not consistent with the definition of **C**. For example, the program $\{\underline{S}_0(x) \leftarrow \neg \underline{S}_1(x); \underline{S}_1(x) \leftarrow \underline{R}(x)\}$ represents the query whose value is $D - R$; however, the following formula (with nonterminals) $(\forall x. \neg \underline{S}_1(x) \rightarrow \underline{S}_0(x)) \wedge (\forall x. R(x) \rightarrow S_1(x))$ represents the query whose value is always $\{\ \}$. Therefore FN does not obviously generalize **C**.

In order to characterize the class FN, we need to define the second-order queries. A universally quantified second-order formula $\Phi(\bar{x})$ has the form $\forall \bar{\underline{S}}.\Psi(\bar{\underline{S}}, \bar{\underline{R}}, \bar{x})$, where Ψ is first order, and it defines the query Q_Φ given by (analogous to first-order queries):

$$Q_\Phi(B) = \{ \bar{d} | \Phi(\bar{d}) \text{ is true in } B \}.$$

The set of such queries is denoted US.

Theorem 5.5. FN = US.

PROOF. To show FN $\subset$ US, let $\Phi(\underline{S}_0, \underline{S}_1, \ldots)$ represent a query Query_Φ in FN. Then let $\Psi(\bar{x})$ denote the formula

$$\forall \underline{S}_0, \underline{S}_1, \ldots (\Phi(\underline{S}_0, \underline{S}_1, \ldots) \rightarrow \underline{S}_0(\bar{x})).$$

It is readily seen that $\text{Query}_\Phi = Q_\Psi$.

To show US $\subset$ FN, let Ψ be a formula $\forall \underline{\bar{S}}.\Psi'(\underline{\bar{S}}, \bar{\underline{R}}, \bar{x})$ that represents a query Q_Ψ in US. Then let $\Phi(\underline{\bar{S}}, \underline{S}_0)$ be defined as follows (here S_0 is a new relation symbol of rank $|\bar{x}|$):

$$\forall \bar{y}.(\Psi'(\underline{\bar{S}}, \bar{\underline{R}}, \bar{y}) \rightarrow \underline{S}_0(\bar{y})).$$

Then it can be seen that $\text{Query}_\Phi = Q_\Psi$. □

We have defined three generalizations of Horn clause queries **H**. These include the extended Horn clause queries **H'**, the clausal queries **C**, and the first-order queries with nonterminals FN. These four classes have been characterized by the query classes YE^+, YE, FP, and US, respectively. The connection between these four is given by

Theorem 5.6. $\text{YE}^+ \subset \neq \text{YE} \subset \neq \text{FP} \subset \neq \text{US}$.

PROOF. We only need to show that FP $\subset \neq$ US. We first show that FP $\subset$ US. It has been shown by Immerman [11] that FP = E $\circ$ YF. In other words, any query Q in FP can be represented by a fixpoint formula $\Psi(\bar{x}')$ with only one fixpoint operator, having the form:

$$\exists \bar{y}'.((\bar{x}, \bar{y}).Y\underline{S})\Phi(\underline{S}, \bar{z}).$$

Here Φ is a first-order formula, and $\bar{x}'$, $\bar{y}'$ are the vectors $\bar{x}$, $\bar{y}$ without repeated variables ($\bar{x}'$, $\bar{y}'$ are disjoint, and $|\bar{x}| + |\bar{y}| = |\bar{z}| =$ rank of S). Q can be represented as a universal second-order query by the formula $\Psi'(\bar{x}')$:

$$\forall \underline{S}.\exists \bar{y}'.((\forall \bar{z}.\Phi(\underline{S}, \bar{z}) \equiv \underline{S}(\bar{z})) \rightarrow \underline{S}(\bar{x}, \bar{y})).$$

Here $\bar{x}$, $\bar{y}$, $\bar{x}'$, $\bar{y}'$ are as above. Now $\Psi'(\bar{x})$ is equivalent to $\Psi(\bar{x})$ since both are equivalent to:

$$\exists \bar{y}'.\forall \underline{S}.((\forall \bar{z}.\Phi(\underline{S}, \bar{z}) \equiv \underline{S}(\bar{z})) \rightarrow \underline{S}(\bar{x}, \bar{y}))$$

because Ψ has a least fixpoint.

To show FP $\neq$ US, consider the query ODD of type $(\,) \rightarrow 1$ whose value on $B = (D, R)$ is:

$$\text{ODD}(B) = \begin{cases} D & \text{if } |D| \text{ is odd} \\ \{\,\} & \text{otherwise.} \end{cases}$$

It has been shown [5] that ODD $\notin$ FP (the query used there was called EVEN, with $\text{EVEN}(B) = \{(\,)\}$ if $|D|$ is even, $\{\,\}$ otherwise: this difference is immaterial since FP is closed under negation and projection). However, ODD is in US. It can be represented by:

$$\forall \underline{S}.(\exists x, y, z.\underline{S}(x, y) \wedge \underline{S}(y, z)) \vee (\exists x.\forall y \neg \underline{S}(x, y)) \vee (\exists x.\forall y.\neg \underline{S}(y, x))$$

(the formula states that S cannot be a perfect matching on the elements of D). □

6. CONCLUSIONS

The purpose of this paper has been to characterize the database queries expressible in the logic programming style. In this style, queries are expressed by programs represented by (restricted) first-order formulas that may contain "nonterminal"

relation symbols. Such non-terminals do not correspond to the given relations of the database, but are rather thought of as "defined" or "programmed" relations. The semantics of these nonterminals is particularly clean when the formulas are restricted to be sets of Horn clauses. The class **H** of such Horn clause programs is shown to express precisely the same queries as YE^+: this is the class of queries expressible using one fixpoint operator applied to positive existential queries.

One limitation of the queries expressed by Horn clause programs is that they are all monotone, i.e., adding tuples to relations in the database cannot cause any tuple in the output of the query to be deleted. This is easily overcome without sacrificing the clean semantics by extending Horn clauses to allow negated terminal relation symbols as well. Such extended Horn programs **H'** express the queries YE containing one fixpoint operator applied to (not necessarily positive) existential queries.

However, it is shown that even **H'** cannot express universal quantification. It therefore does not express all first-order queries, even though it (even **H**) expresses some queries (e.g., the transitive closure query) that are not first order.

In this paper we have considered two (noncompatible) ways of extending **H'** further. One is to allow negated the nonterminals as well, but to impose an order on the evaluation of nonterminals. We argue that this extension is in the same spirit of resolution-based proofs as Horn clause programs, and show that the resulting "clausal programs" **C** express precisely the fixpoint queries FP (queries closed under the first-order operations and fixpoints as well).

The second, and aesthetically more appealing, way of extending **H'** is to allow arbitrary first-order formulas with nonterminals as programs yielding FN. A semantics consistent with **H'** (but not with **C**) is to take the intersection of all solutions for the nonterminals that satisfy the formula. FN is shown to be the same as universally quantified second-order logic (without nonterminals) US, and to contain FP as a strict subset. This semantics is, however, not very desirable from a computational viewpoint, in that it is not apparent how to write efficient programs (or a good compiler) since one has to search all subsets of the data base. This objection could conceivably be overcome by attaching a different semantics to first-order formulas with nonterminals. One possibility here is the following. Let a first-order formula $\Phi(\bar{R}, \underline{S}_0, \underline{S}_1, \ldots)$ represent a query whose value on $B = (D, \bar{R})$ is the set of all $\bar{d}$ such that $\underline{S}_0(\bar{d})$ is provable (in some proof system, e.g., that of first-order logic on finite and infinite domains) from Φ and the literals corresponding to B (if $\bar{e} \in R_i$ (resp. $\bar{e} \notin R_i$) in B, this corresponds to the literal $\underline{R}_i(\bar{e})$ (resp. $\neg \underline{R}_i(\bar{e})$), similarly for $=$, $\neq$). The possibility is in line with the developing area of *deductive databases*; see, e.g., [16].

There are interesting connections between the work presented here and the theory of inductive definability [10, 13]. For instance, the collapsing of mutual fixpoints to a single one is used in our proof of $\mathbf{H} = YE^+$. A similar result for arbitrary (i.e., not necessarily finite) structures appears in [13, Sec. 1C]. Also, the results $\mathbf{H} = YE^+$, $\mathbf{H'} = YE$, $\mathbf{C} = FP$, and $FN = US$ apply to arbitrary structures as well as to finite ones, and the fixpoint queries (FP) are precisely the inductive definitions on *finite* structures.

We would like to thank A. Shamir and M. Vardi for their helpful comments.

REFERENCES

1. Apt, K. R. and van Emden, M. H., Contributions to the Theory of Logic Programming, *J. ACM* 29:841–862 (1982).
2. Aho, A. V. and Ullman, J. D., Universality of Data Retrieval Languages, *6th ACM Symp. on Principles of Programming Languages*, San-Antonio, TX (Jan 1979), pp. 110–117.
3. Clark, K. L., Negation as Failure, in: H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, New York, 1978, pp. 293–322.
4. Coelho, H., Cotta, J. C., and Pereira, L. M., How to Solve it with PROLOG, Report, Laboratório Nacional de Engenharia Civil, Ministério da Habitacao e Obras Publicas, Lisbon, 1980.
5. Chandra, A. K. and Harel, D., Structure and Complexity of Relational Queries, *J. Com. Sys. Sci.* 25:99–128 (1982).
6. van Emden, M. H., Computation and Deductive Information Retrieval, in: E. J. Neuhold (ed.), *Formal Description of Programming Concepts*, North-Holland, New York, 1978, pp. 421–439.
7. van Emden, M. H. and Kowalski, R. A., The Semantics of Predicate Logic as a Programming Language, *J. ACM*, 23:733–742 (1976).
8. Gallaire, H. and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, New York, 1978.
9. Harel, D., Review on *Logic and Data Bases*, *Computing Reviews* #36,671 (Aug 1980), pp. 367–369.
10. Harel, D. and Kozen, D., A Programming Language for the Inductive Sets, and Applications, *Information and Control*, to appear. [Also Proc. ICALP 82, Aarhus, Denmark (July 1982).]
11. Immerman, N., Relational Queries Computable in Polynomial Time, *14th Ann. ACM Symp. on Theory of Computing*. San Francisco, CA, May 1982, pp. 147–152.
12. Kowalski, R., Predicate Logic as a Programming Language, *Proc. IFIP Cong. 1974*, North-Holland, Amsterdam, 1974, pp. 569–574.
13. Moschovakis, Y. N., *Elementary Induction on Abstract Structures*, North-Holland, New York, 1974.
14. Roussel, P., PROLOG: Manuel de Réference et d'Utilisation, Report, Groupe de IA, UER Luminy, Univ. d'Aix-Marseille, France (1975).
15. Clocksin, W. and Mellish, C., *Programming in PROLOG*, Springer-Verlag, Berlin, 1981.
16. Gallaire, H., Minker, J., and Nicolas, J.-M., Logic and Databases: A Deductive Approach, *Computing Surveys* 16: 153–185 (1984).