Relational Queries Computable in Polynomial Time

Extended Abstract

Neil Immerman

Tufts University
Dept. of Mathematics
Medford, Mass. 02155

Laboratory for Computer Science
M.I.T.
Cambridge, Mass. 02139

Introduction and Summary

Query languages for relational databases have received considerable attention. In 1972 Codd [Cod72] showed that two natural mathematical languages for queries -- one algebraic and the other a version of first order predicate calculus -- had identical powers of expressibility. Query languages which are as expressive as Codd's Relational Calculus are sometimes called complete. This term is misleading, however, because many interesting queries are not expressible in "complete" languages.

In 1979, Aho and Ullman [AhU179] noted that relational calculus does not suffice to express the transitive closure property. They suggested adding a least fixpoint operator to relational calculus in order to create a query language which can express transitive closure. In 1980, Chandra and Harel [ChHa80b] studied the expressive power of relational calculus with added primitives such as a least fixpoint operator. They defined a Fixpoint Hierarchy of query classes, the queries in each particular class being those expressible with a certain number of applications of the least fixpoint operator, followed by a certain number of alternations of ordinary quantification. In this paper we show:

Theorem 2: The Fixpoint Hierarchy collapses at the first fixpoint level.

That is, any query expressible with several applications of least fixpoint can already be expressed with one. We also show:

Theorem 1: Let L be a query language consisting of relational calculus plus the least fixpoint operator. Suppose that L contains a relation symbol for a total ordering relation on the domain (e.g. lexicographic ordering). Then the queries expressible in L are exactly the queries computable in polynomial time.

Theorem 1 was discovered independantly by M. Vardi [Var82]. It gives a simple syntactic categorization of those queries which can be answered in polynomial time. Of course queries requiring polynomial time in the size of the database are usually prohibitatively expensive. We also consider weaker languages for expressing less complex queries.

Section 1: Background and Notation

This section will briefly define and give examples of the objects under consideration. The reader is referred to [Ull80], [End72], and [AHU74] for excellent discussions of relational query languages, first order predicate calculus, and computational complexity, respectively.

First, a relational database $B = \langle D, R_1 \ldots R_k \rangle$ consists of a finite domain $D = \{e_1 \ldots e_n\}$, and a finite set of relations on the domain. R_i is an a_i-ary relation on D, i.e. $R_i \subseteq D^{a_i}$

As an example consider the database:

$$B_0 = \langle D_0, \text{FEMALE}, \text{PARENT}, \text{HUSBAND} \rangle$$

consisting of a domain of persons:

$$D_0 = \{\text{Abraham, Isaac, Sarah, Leah, Rebekah, Jacob, Rachel, Joseph, Benjamin} \ldots\}$$

B_0 has a monadic relation, FEMALE, true of the female members of the domain, i.e.

$$\text{FEMALE} = \{\text{Sarah, Leah, Rebekah, Rachel}, \ldots\}$$

and two binary relations, PARENT(x,y) true when x is a parent of y, and HUSBAND(x,y) true when x is the husband of y. Thus,

$$\text{PARENT} = \{\langle\text{Abraham,Isaac}\rangle, \langle\text{Sarah,Isaac}\rangle, \langle\text{Isaac,Jacob}\rangle, \langle\text{Rebekah,Jacob}\rangle, \langle\text{Jacob,Joseph}\rangle, \langle\text{Jacob,Benjamin}\rangle, \langle\text{Rachel,Joseph}\rangle, \langle\text{Rachel,Benjamin}\rangle, \ldots\}$$

$$\text{HUSBAND} = \{\langle\text{Abraham,Sarah}\rangle, \langle\text{Jacob,Leah}\rangle, \langle\text{Jacob,Rachel}\rangle, \langle\text{Isaac,Rebekah}\rangle, \ldots\}$$

A relational scheme $\langle R_1 \ldots R_k \rangle$ is just a finite list of relation symbols. R_i is an a_i-ary relation symbol. For example B_0 is an instance of the relational scheme FAMILY = <FEMALE, PARENT, HUSBAND>, where "FEMALE" is a monadic relation symbol, and "PARENT" and "HUSBAND" are binary relation symbols. Generally, the difference between relations and relation symbols will be determined by context -- to be rigorous we should give the actual relations in B_0 the superscript "B_0".

We can now define Domain Relational Calculus, a query language based on first order predicate calculus. If $S = \langle R_1 \ldots R_k \rangle$ is any relational scheme then L(S), the relational calculus language of S, is built up from the following:

Relation Symbols:	$R_1, \ldots, R_k, =$
Logical Connectives:	$\wedge, \vee, \neg$
Variables:	$x, y, z, \ldots$
Quantifiers:	$(\forall x), (\exists x)$

Well formed formulas (WFF's) are constructed using the above symbols in the usual way. For example, we can express the sibling relation by the formula:

$$\mathrm{Sib}(u,v) \equiv \exists x \exists y \big(x \neq y \wedge \mathrm{PARENT}(x,u) \wedge \mathrm{PARENT}(x,v) \wedge u \neq v \wedge \mathrm{PARENT}(y,u) \wedge \mathrm{PARENT}(y,v) \big)$$

Sib is a WFF in L(FAMILY) with two free variables, u and v. It can be thought of as a query to a FAMILY database B. The answer would be the set of pairs $\langle c_1,c_2 \rangle$ from D such that B satisfies $\mathrm{Sib}(c_1,c_2)$. For example, B_0's response would include the pair, <Benjamin,Joseph>, because B_0 satisfies Sib(Benjamin,Joseph) -- in symbols, $B_0 \models \mathrm{Sib(Benjamin,Joseph)}$. Note that any instance B of the relational scheme S "understands" any formula from L(S) because B has a relation corresponding to each relation symbol in S.

The reader should convince himself or herself that many queries can be expressed in relational calculus. As further examples, we write the expressions for second cousin, and for second cousin once removed (Scor):

$$2^{nd}\mathrm{Cos}(u,v) \equiv (\exists\, x\, y\, z\, w)\big(\mathrm{Sib}(z,w) \wedge \mathrm{PARENT}(z,x) \wedge \mathrm{PARENT}(w,y) \wedge \mathrm{PARENT}(x,u) \wedge \mathrm{PARENT}(y,v)\big)$$

$$\mathrm{Scor}(w,z) \equiv (\exists x)\big([\mathrm{PARENT}(x,w) \wedge 2^{nd}\mathrm{Cos}(x,b)] \vee [\mathrm{PARENT}(x,z) \wedge 2^{nd}\mathrm{Cos}(x,w)] \big)$$

Section 2: Adding a Least Fixpoint Operator

Relational Calculus corresponds exactly to the familiar notion of first order predicate logic. This language forms a rich class of queries. Of course not all properties one might want to ask a database about are first order expressible. In the Relational Calculus for our FAMILY scheme it is impossible to express the relation Ancestor(x,y). In [AhUl79] Aho and Ullman suggest adding a least fixpoint operator to relational calculus so that transitive closures such as Ancestor(x,y) may be expressed.

For example, consider the following first order expression:

$$\varphi_A(R)[x,y] \equiv \big(x=y \vee \exists z[\mathrm{PARENT}(x,z) \wedge R(z,y)]\big)$$

For any FAMILY database, B, φ_A maps each binary relation, R_1, on the domain of B to the binary relation:

$$\varphi_A(R_1) \equiv \{ \langle x,y \rangle \mid B \models \varphi_A(R_1)[x,y] \}$$

φ_A is monotone, i.e, $R_1 \subseteq R_2$ implies $\varphi_A(R_1) \subseteq \varphi_A(R_2)$. Thus for any database B, φ_A has a least fixpoint, i.e. a relation R_0 such that $\varphi_A(R_0)=R_0$ and R_0 is minimal with this property. It is well known that an expression, $\varphi(R)$, is monotone iff it is equivalent to an expression $\varphi'(R)$ in which R occurs only positively. Following [AhUl79] we will use a least fixpoint operator, LFP, on monotone expressions such as φ_A. It is easy to see, for example, that

$$\mathrm{Ancestor} = \mathrm{LFP}(\varphi_A)$$

It is interesting to consider the computational complexity of evaluating queries that use this least fixpoint operator. The following proposition is due to Chandra and Harel, [ChHa80a].

Proposition 1: Given a database, $B=\langle D\ R_1 \ldots R_k \rangle$, and a monotone operator φ in the language of B, LFP(φ) exists and is computable in time p(|D|) for some polynomial p.

proof: Let n=|D|, the size of the domain, and let a be the arity of φ. Define $R_0 = \varphi^{[n^a]}(\emptyset)$, i.e. R_0 is the relation resulting from composing φ with itself n^a times and applying it to the empty set. Obviously $R_0=\mathrm{LFP}(\varphi)$ because each application of φ adds some tuples to the at most n^a tuples in the relation. No additional tuples can be added after n^a steps. Let

$$\varphi(R)[x_1 \ldots x_a] \equiv (Q_1 z_1 \ldots Q_k z_k)\ M(x_1 \ldots x_a, z_1 \ldots z_k, R)$$

where M is quantifier free. Given R as a list of tuples we can compute $\varphi(R)$ in time $n^{a+k}\log(n)$ by cycling through all possible values of $z_1 \ldots z_k$ for each possible value of x. Iterating φ n^a times, we can compute LFP(φ) in time $n^{2a+k+1} = p(n)$. ∎

Chandra and Harel have considered a Fixpoint Hierarchy, FP, which consists of alternating applications of quantification and LFP. Inductively:

$$\Sigma_0 = \Pi_0 = \{ M \mid M \text{ is a quantifier free query.} \}$$
$$\Sigma_{\alpha+1} = \{(\exists x)\psi(x) \mid \psi\ \varepsilon\ \Pi_\alpha \}$$
$$\Pi_\alpha = \{ \neg\varphi \mid \varphi\ \varepsilon\ \Sigma_\alpha \}$$

For β a limit ordinal,

$$\Sigma_\beta = \{ \psi(v) \mid \psi = \mathrm{LFP}(\varphi), \varphi \text{ monotone}, \varphi\ \varepsilon\ \Sigma_\alpha, \alpha < \beta \}$$

Thus Σ_n is the set of queries expressible with n alternations of quantification beginning with existential. $\Sigma_{\omega n}$ is the set of queries expressible using n applications of LFP with intermediate applications of quantification and negation. The last line defines Σ_β for β a limit ordinal, as a substitution of variables, v, into the least fixpoint of a formula φ which is lower down in the hierarchy. N.B. we have slightly modified the definition in [ChHa80] which did not allow such substitutions.

It is known that additional alternations of first order quantification give increased expressibility and that transitive closure is not first order expressible. See [ChHa80] and [AhUl79]. Thus:

Fact: The Fixpoint Hierarchy is strict up to Σ_ω, that is, the following containments are all strict:

$$\Sigma_0 \subset \Sigma_1 \subset \Sigma_2 \subset \quad \ldots \subset \quad \Sigma_\omega$$

Chandra and Harel ask whether the hierarchy continues past Σ_ω and we will show in Theorem 2 that it does not. They also considered the computational complexity of answering queries. Let QPTIME be the set of queries computable in polynomial time in the size of the database:

$$\text{QPTIME} \quad \equiv \quad \{ \ \varphi \ | \ \text{Graph}(\varphi) \in P \ \}$$

where $\text{Graph}(\varphi) = \{\langle B,c\rangle \mid B \models \varphi(c)\}$. We have already seen in Proposition 1 that Σ_ω is contained in QPTIME. It follows that FP $\subseteq$ QPTIME. Chandra and Harel show that equality does not hold.

Theorem[ChHa80b]: FP $\neq$ QPTIME

The proof has to do with the fact that queries in FP don't necessarily have the ability to count. Thus for example the query concerning family databases, "Is there an even number of females?", is not expressible in the Fixpoint Hierarchy.

The inability of our queries to count can be eliminated by adding to the language an ordering of the domain. Such an ordering, e.g. by bit representation, is always available in real databases. Let Q($\leq$)PTIME be the set of queries computable in polynomial time on ordered databases. That is we only consider databases which have a total order, $\leq$, on the domain. Let FP($\leq$) be the queries in FP, where $\leq$ must be interpretated as a total ordering of the domain. We show in the next section that Q($\leq$)PTIME = FP($\leq$), and in fact only one application of LFP is needed.

Section 3: Main Results

Theorem 1: Q($\leq$)PTIME $=$ $\Sigma_\omega(\leq)$

proof sketch: That Q($\leq$)PTIME contains $\Sigma_\omega(\leq)$ is clear. We must show the converse. Let $S = \{\langle B_1,c_1\rangle \ . \ . \ . \ \}$ be a set of pairs of ordered databases B belonging to a certain relational scheme and r-tuples, c, from B. Let M be a Turing machine that accepts S in time n^k. We must show that there is a query $\alpha(x_1...x_r) \in \Sigma_\omega$ which expresses S: i.e.

$$S \quad = \quad \{ \ \langle B,c\rangle \ | \ B \models \alpha(c) \ \} \ .$$

Each candidate for S is a pair, $\langle B,c\rangle$, where B has an n element domain, D, with a total ordering, $\leq$, on its elements. Thus we can think of D as the set of integers from 0 to n-1. We can use k-tuples of variables to denote numbers between 0 and n^k-1. We will use one application of LFP to write the query $\text{Cell}(x_1...x_r,p_1...p_k,t_1...t_k,a)$ to express the statement that in M's computation the contents of cell number $p_1...p_k$ at time $t_1...t_k$ is a. More precisely, we will show that there is a first order sentence $\psi(S)[x_1...x_r,p_1...p_k,t_1...t_k,a]$ such that Cell = LFP(ψ), and that B satisfies Cell(c,p,t,a) if and only if the instantaneous description of M's computation on input $\langle B,c\rangle$ contains symbol a in cell **p** at time **t**.

Once we have Cell we can let $\alpha(x) \equiv \text{Cell}(x,0,n^k\text{-}1,q_f)$. Here α says that M is in its accept state, q_f, after n^k-1 steps. Thus, as desired,

$$B \models \alpha(c) \qquad \leftrightarrow \qquad \langle B,c\rangle \in S \ .$$

There are two steps to writing ψ whose least fixpoint is Cell. First we must write the sentence $M_0(c,p,a)$ meaning that at time 0, cell **p** is a, i.e. that the initial input tape contains $\langle B,c\rangle$. Suppose that $B = \langle\{0...n\text{-}1\},R_1...R_s\rangle$. Then the input will consist of an n^{a_1} bit table for R_1, followed by an n^{a_2} bit table for R_2, and so on, followed by some binary representation for $c_1 ... c_r$. It is easy to see that using $\leq$ and relation symbols R_1 ... R_k we can write the first order sentence M_0 saying that the input is correct.

Consider the following monotone first order expression ψ:

$$\begin{aligned}\psi(S)[x,p,t,a] \quad \equiv \quad & \Big(t=0 \wedge M_0(x,p,a)\Big) \quad \vee \\ & \exists a_{-1}a_0a_1\Big(\langle a_{-1}a_0a_1\rangle \rightarrow a \wedge S(x,p\text{-}1,t\text{-}1,a_{-1}) \\ & \wedge \quad S(x,p,t\text{-}1,a_0) \wedge S(x,p+1,t\text{-}1,a_1)\Big)\end{aligned}$$

Here "$\langle a_{-1}a_0a_1\rangle \rightarrow a$" means that the triple $\langle a_{-1}a_0a_1\rangle$ leads to the symbol a in one move of M. Thus each application of ψ gives us one more row of M's computation, so LFP(ψ) = Cell. To write $\alpha(x) = \text{Cell}(x,0,n^k\text{-}1,q_r)$ would seem to require some quantification after the least fixpoint operator is used. In fact we can add an extra variable, z, to ψ so that it does two things: case 1 ($z \neq x_1$): compute Cell; case 2 ($z = x_1$): check if $\text{Cell}(x,0,n^k\text{-}1,q_f)$ holds. Thus:

$$\begin{aligned}\Psi(S)[z,x,p,t,a] \equiv & \Big(z\neq x_1 \wedge \psi(S)[x,p,t,a]\Big) \vee \Big(z=x_1 \quad \wedge \\ & (\exists z'\,uva')\Big[z'\neq x_1 \wedge u=0 \wedge v=n\text{-}1 \\ & \wedge a'=q_f \wedge S(z',x,u...u,v...v,a')\Big]\Big)\end{aligned}$$

Let S_0 = LFP(Ψ), and let $\alpha(x_1...x_r) = S_0(x_1,x_1...x_r,x_1...x_1)$. Then α is equivalent to $\text{Cell}(x_1...x_r,0,n^k\text{-}1,q_f)$ as desired. ∎

If we do not have an ordering then it is not possible in general to simulate a computation. However we can show that the hierarchy still collapses at the first fixpoint level:

Theorem 2: FP $=$ Σ_ω .

We first give the proof for an example:

Claim: $\neg$Ancestor $\in \Sigma_\omega$.

proof: Recall the formula φ_A such that LFP(φ_A) = Ancestor:

$$\varphi_A(R)[x,y] \quad \equiv \ (x=y) \ \vee \ \exists z(\text{Par}(x,z) \ \wedge \ R(z,y))$$

Our problem is to monotonically add information about ancestors to some larger relation S so that after finitely many steps we will be able to tell that we are done, and for which values Ancestor(x,y) does not hold. Define rank(x,y) = min k : $\varphi_A^{(k)}(\varnothing)[x,y]$. That is, rank(x,y) is the minimum number of applications of φ_A needed to discover that Ancestor(x,y) holds. Clearly rank(x,y) = distance(x,y)+1. Even though our language does not have access to numbers we can use LFP to make statements about rank. We will write first order expressions φ_1, φ_2, φ_3, with least fixpoints GE, GT, GS, respectively, meaning the following:

$$GE(x,y,u,v) \equiv (\,rank(u,v) < \infty \;\wedge\; rank(x,y) \geq rank(u,v)\,)$$

$$GT(x,y,u,v) \equiv (\,rank(u,v) < \infty \;\wedge\; rank(x,y) > rank(u,v)\,)$$

$$GS(x,y,u,v) \equiv (\,rank(u,v) < \infty \;\wedge\; rank(x,y) > rank(u,v)+1)$$

Once we have these three predicates we can express $\neg$Ancestor without using negation. First we can write Diam(a,b) expressing the property that distance(a,b) is the maximum possible.

$$\begin{aligned} Diam(a,b) \equiv\; & GE(a,b,a,b) \wedge (\forall uv)\Big[GE(a,b,u,v) \\ & \vee\; (\,GT(u,v,a,b) \wedge GS(u,v,a,b)\,)\Big] \end{aligned}$$

Diam(a,b) says that rank(a,b) is finite and that no pair u,v satisfies rank(u,v) = rank(a,b) + 1. Now the pairs of infinite rank are just those pairs with rank strictly greater than that of a diameter:

$$\neg Ancestor(x,y) \equiv \exists ab(Diam(a,b) \wedge GT(x,y,a,b))$$

Here are the formulas, φ_1, φ_2, and φ_3, used to define GE, GT, and GS:

$$\begin{aligned} \varphi_1(R_1)[x,y,u,v] \equiv\; & (u=v) \;\vee \\ & \Big[\neg(x=y) \wedge (\exists u'\, \forall x')\Big((Par(u,u') \wedge \\ & [\neg(x=x' \vee Par(x,x')) \vee R_1(x',y,u',v)]\Big)\Big] \end{aligned}$$

$$\begin{aligned} \varphi_2(R_2)[x,y,u,v] \equiv\; & (u=v \wedge x\neq y) \;\vee \\ & \Big[\neg(x=y \vee Par(x,y)) \wedge (\exists u'\, \forall x')\Big((Par(u,u') \\ & \wedge\; [\neg(x=x' \vee Par(x,x') \vee R_2(x',y,u',v)\,]\Big)\Big] \end{aligned}$$

$$\begin{aligned} \varphi_3(R_3)[x,y,u,v] \equiv\; & (u=v \wedge x\neq y \wedge \neg Par(x,y)) \vee \\ & \Big[\neg\Big(x=y \vee Par(x,y) \vee \exists z(Par(x,z) \wedge Par(z,y)\Big) \wedge \\ & (\exists u'x')\Big((Par(u,u') \wedge [\neg(x=x' \vee Par(x,x') \vee R_3(x',y,u',v)]\Big)\Big] \end{aligned}$$

It is easy to verify that GE = LFP(φ_1), GT = LFP(φ_2), and GS = LFP(φ_3). We will be done once we show how to combine φ_1, φ_2, φ_3 into one simultaneous LFP:

$$\begin{aligned} \Phi(S)[z_1,z_2,z_3,u,v,x,y] \equiv\; & \Big((z_1\neq z_2 \wedge z_2\neq z_3) \wedge \varphi_1(S)[u,v,x,y]\Big) \\ \vee\; & \Big((z_1\neq z_2 \wedge z_2=z_3) \wedge \varphi_2(S)[u,v,x,y]\Big) \\ \vee\; & \Big((z_1=z_2 \wedge z_2\neq z_3) \wedge \varphi_3(S)[u,v,x,y]\Big) \\ \vee\; & \Big((z_1=z_2 \wedge z_2=z_3) \wedge (\exists a\, b)[Diam'(a,b) \\ & \wedge\; S(a,b,b,x,y,a,b)]\Big) \end{aligned}$$

Φ uses the three variables, z_1,z_2,z_3, to break the definition into four cases. We are assuming that the domain is of size at least two and that the diameter is nonzero. Let Diam'(a,b) be Diam(a,b) with S(a,b,a,u,v,s,y), S(a,b,b,u,v,x,y), S(a,a,b,u,v,x,y) substituted for GE, GT, and GS, respectively. Thus Diam'(a,b) will hold only when the fixpoint has been reached and a and b are of maximal distance. In this case S(a,b,b,x,y,a,b) is equivalent to GT(x,y,a,b). Thus letting S_0 = LFP(Φ), we have

$$\neg Ancestor(x,y) \equiv S_0(x,x,x,x,x,x,y)\ .$$

This proves the claim. ∎

We now sketch the general proof of Theorem 2. Let R_0 = LFP(φ) be an arbitrary least fixpoint of arity r. As above we define the <u>rank(x)</u> to be the minimum k such that $\varphi^{(k)}(\emptyset)[x]$ holds. Then as in the above example we can define the relations, GE, GT, and GS, as simultaneous fixpoints where

$$GE(x,u) \equiv (\,rank(u) < \infty \;\wedge\; rank(x) \geq rank(u)\,)$$

$$GT(x,u) \equiv (\,rank(u) < \infty \;\wedge\; rank(x) > rank(u)\,)$$

$$GS(x,u) \equiv (\,rank(u) < \infty \;\wedge\; rank(x) > rank(u)+1)$$

We can compute R_0 along with its negation in a single fixpoint expression of arity 2r +3. Further steps of quantification and even other fixpoints can then be embedded in the last case of the definition of Φ, as above. Using the next two lemmas we sketch the construction of φ_1, φ_2, and φ_3 whose least fixpoints are GE, GT, and GS, respectively.

<u>Lemma 2.1</u>: Suppose that R is an r-ary relation symbol and that $\varphi(R)[x_1 \ldots x_r]$ is monotonic in R. Then φ may be expressed in an equivalent form:

$$\varphi(R)[x_1 \ldots x_r] \equiv (Q_1 z_1 \,.\, M_1)\ldots(Q_t\, z_t \,.\, M_t)(\exists x_1 \ldots x_r \,.\, M_{t+1})R(x_1 \ldots x_r)$$

where $M_1 \ldots M_{t+1}$ are quantifier free and contain no occurrences of R. Here, $(\forall z\,.\,M)P$ means $(\forall z)(M \rightarrow P)$, and $(\exists z\,.\,M)P$ means $(\exists z)(M \wedge P)$.

<u>example</u>: Let

$$\varphi_B(R)[x_1,x_2] \equiv (x_1=x_2 \vee Par(x_1,x_2)) \vee \exists z[R(x_1,z) \wedge R(z,x_2)]$$

We can express φ_B in the above form as

$$\varphi_B(R)[x_1,x_2] \equiv (\forall z \,.\, M_1)(\exists z)(\forall u\ v \,.\, M_2)(\exists x_1 x_2 \,.\, M_3)R(x_1,x_2)$$

where $M_1 \equiv \neg[x_1=x_2 \vee Par(x_1,x_2)]$

$$M_2 \equiv [(u=x_1 \wedge v=z) \vee (u=z \wedge v=x_2)]$$

$$M_3 \equiv [x_1=u \wedge x_2=v]$$

I hope that the reader will convince herself or himself that we have indeed expressed φ_B in the correct form. The proof of Lemma 2.1 would be by induction on the complexity of φ. Note our use of an abbreviation trick -- the universal quantification of u and v, above, reduced the number of occurrences of R. See Lemma A.3 in [Imm82] for a similar result proved in some detail.

Lemma 2.1 shows how to write any monotonic expression, $\varphi(R)$, in a very simple form. For any such φ, the following lemma shows how to write φ_1, φ_2, φ_3 whose fixpoints are GE, GS, and GT, respectively. The proof of Theorem 2 then follows exactly as in the above example for $\neg$Ancester.

Lemma 2.2: Supose $\varphi(R)[x] \equiv QB(x)[R(x)]$ is in the form of Lemma 2.1, where

$$QB(x) \equiv (Q_1 z_1 . M_1) \dots (Q_t z_t . M_t)(\exists x . M_{t+1})$$

Let $QB(u) \equiv QB(u,z'/x,z)$ i.e. $QB(x)$ with u_i substituted for x_i, $i=1...r$, and z_j' substituted for z_j, $j=1...t$.

Let $\overline{QB}$ be QB with all $\forall$'s replace by $\exists$'s and vice versa. Put

$$\varphi_1(R_1)[x,u] \equiv \varphi(\emptyset)[u] \vee R_1[x,u] \vee \left[QB(u)\overline{QB(x)}\right](R_1[x,u])$$

$$\varphi_2(R_2)[x,u] \equiv (\varphi(\emptyset)[u] \wedge \neg\varphi(\emptyset)[x]) \vee R_2[x,u] \vee \left[QB(u)\overline{QB(x)}\right](R_2[x,u])$$

$$\varphi_3(R_3)[x,u] \equiv (\varphi(\emptyset)[u]\neg\varphi(\emptyset)[x]\wedge\neg\varphi^{(2)}(\emptyset)[x]) \vee R_2[x,u] \vee \left[QB(u)\overline{QB(x)}\right](R_3[x,u])$$

then:

(a): $LFP(\varphi_1) = GE$

(b): $LFP(\varphi_2) = GT$

(c): $LFP(\varphi_3) = GS$

proof of a: By induction on k we show that

$$\varphi_1^{(k)}(\emptyset)[x,u] \leftrightarrow (\,rank(u)\leq k \wedge rank(x)\geq rank(u)\,) \qquad (*)$$

This is clear if $k=1$. Assume that (*) holds for k and consider the following assertion:

$$(\,rank(u^0) \leq k+1 \;\wedge\; rank(x^0)\geq rank(u^0)\,) \qquad (1)$$

This holds iff either of the following conditions is true:

$$(\,rank(u^0) \leq k \;\wedge\; rank(x^0)\geq rank(u^0)\,) \qquad (2)$$

$$(\,rank(u^0) \leq k+1 \;\wedge\; rank(x^0) > k\,) \qquad (3)$$

By induction equation (2) is equivalent to $\varphi_1^{(k)}(\emptyset)[x^0,u^0]$. Equation (3) is equivalent to:

$$\varphi^{(k+1)}(\emptyset)[u^0] \;\wedge\; \neg\varphi^{(k)}(\emptyset)[x^0]$$

Which in turn is equivalent to:

$$\left[QB(u^0)\right]^{(k+1)}(\text{false}) \;\wedge\; \left[\overline{QB(x^0)}\right]^{(k)}(\text{true}) \qquad (4)$$

Now, since the variables in $QB(u^0)$ and $QB(x^0)$ do not intersect, we may transform equation (4) to:

$$QB(u^0)\overline{QB(x^0)}\Big(\left[QB(u)\right]^{(k)}(\text{false}) \;\wedge\; \left[\overline{QB(x)}\right]^{(k-1)}(\text{true})\Big)$$

Or, in other symbols:

$$QB(u^0)\overline{QB(x^0)}\Big(\,rank(u)\leq k \wedge rank(x)> k-1\Big) \qquad (5)$$

We have shown that equations (3) and (5) are equivalent. A consequence of (5) is:

$$QB(u^0)\overline{QB(x^0)}\Big(\,rank(u)\leq k \wedge rank(x)\geq rank(u)\Big) \qquad (6)$$

Which, by the inductive assumption, is equivalent to:

$$QB(u^0)\overline{QB(x^0)}\Big(\varphi_1^{(k)}(\emptyset)[x,u]\Big)$$

Thus, the disjunction of (2) and (3) implies:

$$\varphi_1^{(k)}(\emptyset)[x^0,u^0] \vee QB(u^0)\overline{QB(x^0)}\Big(\varphi_1^{(k)}(\emptyset)[x,u]\Big) \qquad (7)$$

and so $\varphi_1^{(k+1)}(\emptyset)[x^0,u^0]$ holds. In a like fashion we can reverse the steps from (7) to (1), thus proving our claim, (*). It follows that $LFP(\varphi_1) \equiv GE$. The proofs of (b) and (c) are similar. This completes the proof of lemma 2.2 and of the proof sketch of Theorem 2. ∎

Section 4: Conclusions and Directions for Future Work.

Another view of least fixpoint is as an iteration operator. For $\varphi(R)[x_1...x_k]$ monotone, the least fixpoint of φ is just φ iterated n^k times, i.e. $LFP(\varphi) \equiv \varphi^{(n^k)}$. We propose a new query hierarchy which limits this iteration. Define $IQ[f(n)]$ to be the set of queries expressible by iterating a first order query f(n) times to obtain a fixpoint, i.e.

$$IQ[f(n)] \equiv \{\, R_0(v) \mid R_0 = \varphi^{(f(n))}(\emptyset) = \varphi^{(f(n)+1)}(\emptyset)\,\}$$

This definition makes sense for non-monotone φ in which case the iteration could proceed for more than n^k steps and the fixpoint computed need not be the minimal one.

As an example, let

$$\varphi_T(R)[x,y] \equiv \Big(x=y \vee Par(x,y) \vee (\exists z)[R(x,z) \wedge R(z,y)]\Big)$$

It is easy to see that Ancestor $= LFP(\varphi_T) = \varphi^{(\log n)}(\emptyset)$. Thus Ancestor is in IQ[log n]. The following theorem states some results about IQ and $IQ(\leq)$, but many questions, such as, "Is there a strict hierarchy for $IQ(\leq)$?" remain unknown.

Theorem 3:

(a): $IQ[1] =$ First Order Queries $\subsetneq$ QSPACE[log n]

(b): $IQ[\log n] \supseteq$ Transitive Closures

(c): $IQ(\leq)[n^k] =$ FP $=$ QPTIME

(d): $IQ(\leq) =$ PSPACE

Another issue raised by Chandra and Harel, among others, is that languages with an ordering such as $FP(\leq)$ treat differently numbered isomorphic databases differently. That is, the answer to some queries will depend on the ordering. It would be nice to have a language rich enough to simulate Turing machines and yet without this difficulty. One suggestion is to add variables, i j . . . , whose range for a database of size n will be the integers 1 . . . n. We would also add the natural ordering, $\leq$, on these number variables plus "counting quantifiers": $(\exists$ i x's)P(x), meaning, "There exist i distinct x's such that P(x)." These counting quantifiers would remove the most obvious counterexamples to the equality of $IQ(n^k)$ and QPTIME. They are also no stronger than adding $\leq$. More precisely:

Proposition: For $f(n) \geq \log(n)$,

$$IQ(count)[f(n)] \quad \subseteq \quad IQ(\leq)[f(n)] \ .$$

We have shown that all queries using first order quantification and a least fixpoint operator may be expresssed with a single occurrence of least fixpoint applied to a first order expression. Furthermore, in the presence of a total ordering, $\leq$, the queries so expressible are exactly the PTIME computable queries. Finally, a further study of the number of iterations needed to compute fixpoints is desirable. The following open problems merit investigation:

1. Find classes of query languages whose complexity is clear from the syntax, as $FP(\leq)$=PTIME, but for feasible complexity classes from the database point of view, e.g. TIME[n] and $SPACE[\log^k(n)]$.

2. Design Query languages using iterated queries. Study expressibility, complexity and optimization.

3. Prove Hierarchy theorems for:
   (a) IQ[f(n)]
   (b) IQ(count)[f(n)]
   (c) $IQ(\leq)[f(n)]$

4. Prove or disprove:

$$IQ(count)[n^k] = IQ(\leq)[n^k]$$

5. David Harel and Haim Gaiffman have both obtained some results concerning fixpoint hierarchies when the arity of the fixpoints are bounded, [Har81]. Studying the arity of fixpoints is also related to measuring the number of distinct variables used in first order expressions, a problem considered in [Imm82]. It would be interesting and fruitful to study the "expressibility resource" arity of fixpoints in conjunction with others resources such as number of fixpoints or number of iterations.

Acknowledgments

I would like to thank David Harel, John Mitchel, Adi Shamir, and Venkataraman for their helpful ideas concerning this paper. This work was parially supported by NSF grant number MCS 8105754.

Bibliography

[AHU74]: Aho,A., Hopcroft,J., Ullman,J., The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[AhUl79]: Aho,A., Ullman,J.D., "Universality of Data Retrieval Languages," 6th Symp. on Principles of Programming Languages, 1979, pp. 110-117.

[Cha81]: Chandra,A., "Programming Primitives for Database Languages," 8th Symp. on Principles of Programming Languages, 1981, pp. 50-62.

[ChHa80a]: Chandra,A., Harel,D.,"Computable Queries for Relational Databases," JCSS, Vol. 21, No. 2, October, 1980, 156-178.

[ChHa80b]: Chandra,A., Harel,D., "Structure and Complexity of Relational Queries," 21st Symp. on Foundations of Computer Science, 1980, pp: 337-347.

[ChHa82]: Chandra,A.,Harel,D.,"Horn Clause and the Fixpoint Query Hierarchy," to appear.

[Cod72]: Codd,E.F., "Relational Completeness of Database Sublanguages," in Database Systems, R.Rustin, ed., Prentice-Hall, 1972, pp. 65-98.

[End72]: Enderton,H., A Mathematical Introduction to Logic, Academic Press, 1972.

[Har81]: Harel, David, personal communication.

[Imm80]: Immerman,N., "Upper and Lower Bounds for First Order Expressibility," 21st Symp. on Foundations of Computer Science, 1980, pp. 74-82.

[Imm81]: Immerman,N., "Number of Quantifiers is Better than Number of Tape Cells," JCSS, Vol. 22, No. 3, June, 1981, pp. 384-406.

[Imm82]: Immerman,N., "Upper and Lower Bounds for First Order Expressibility," to appear in JCSS, 1982.

[Ull80]: Ullman,J.D., Introduction to Database Systems, Computer Science Press, 1980.

[Var82]: Vardi,M.,"Complexity of Relational Query Languages," this volume.