The Complexity of Relational Query Languages (Extended Abstract)

Conference Paper · January 1982

DOI: 10.1145/800070.802186 · Source: DBLP

1 author:

Moshe Vardi
Rice University
783 PUBLICATIONS **31,173** CITATIONS

SEE PROFILE

THE COMPLEXITY OF RELATIONAL QUERY LANGUAGES

Extended Abstract

Moshe Y. Vardi†

Department of Computer Science
Stanford University
Stanford, California 94305

Abstract

Two complexity measures for query languages are proposed. *Data complexity* is the complexity of evaluating a query in the language as a function of the size of the database, and *expression complexity* is the complexity of evaluating a query in the language as a function of the size of the expression defining the query. We study the data and expression complexity of logical languages - relational calculus and its extensions by transitive closure, fixpoint and second order existential quantification - and algebraic languages - relational algebra and its extensions by bounded and unbounded looping. The pattern which will be shown is that the expression complexity of the investigated languages is one exponential higher then their data complexity, and for both types of complexity we show completeness in some complexity class.

† Research supported by a Weizmann Post-doctoral Fellowship, Fullbright Award, and NSF grant MCS-80-12907. Part of this research was carried out while the author was as the Hebrew University of Jerusalem, Israel, and was supported by Grant 1849/79 of the U.S.A.-Israel BSF.

1. Introduction

In the last years there has been a lot of interest in query languages for relational databases. Following Codd's pioneering work [Codd] on the relational calculus and algebra, a lot of work has been done on studying and comparing the expressive power of several query languages [AU,Ba,CH1,CH2,CH3,Chan,Coop,Pa]. The approach taken here is to compare query languages by investigating the complexity of evaluating queries in these languages.

There are three ways to measure the complexity of evaluating queries in a specific language. First, one can fix a specific query in the language and study the complexity of applying this query to arbitrary databases. The complexity is then given as a function of the size of the databases. We call this complexity *data complexity*.

Alternatively, one can fix a specific database and study the complexity of applying queries represented by arbitrary expressions in the language. The complexity is then given as a function of the length of the expressions. We call this complexity *expression complexity*.

Finally, one can study the complexity of applying queries represented by arbitrary expressions in the language to arbitrary databases. The complexity is then given as a function of the combined size of the expressions and the databases. We call this complexity *combined complexity*.

It turns out that combined complexity is pretty closed to expression complexity, and for this reason we

concentrate in this paper upon data and expression complexity. These two types of complexity actually measures two different things. Data complexity is really a measure for the expressive power of the language because it answers the question "how difficult are the individual questions asked in this language?". It is of interest mainly to users who tend to use only very specific queries. Expression complexity on the other hand is a measure for the succinctness of the language because it answers the question "how difficult is answering different questions asked in this language?". It is of interest mainly to users who tend to use a variety of queries.

There are two major types of relational query languages. Logical languages, e.g., relational calculus, consist of formulas that when applied to a database return as an answer the set of all tuples that satisfy them. These languages are non-procedural in nature. Algebraic languages, e.g., relational algebra, consist of programs whose basic operations are algebraic ones like join and projection. These languages are procedural in nature.

The logical languages investigated in this paper are the language of first-order logic (which differs from the relational calculus of [Codd] in having the variables ranging over domain elements instead of tuples), and the languages obtained by restricting it to be quantifier-free or by enriching it with transitive closure, fixpoint and second-order quantification, respectively. The algebraic languages are the relational algebra and the languages obtained by restricting it to be projection-free or by enriching it with bounded looping and unbounded looping, respectively.

As an example consider the language of first-order logic. Let φ be a sentence in this language of size s (a sentence represents a Boolean query, that is, a query that returns a yes/no answer). φ has at most s variables. In order to evaluate φ on a database of size n, it suffices to cycle through at most n^s possible assignments of values from the database to the variables, and this can be done in space $O(\log n)$. Thus, the set of all databases satisfying φ is in LOGSPACE. On the other hand, if B is a non-trivial database, i.e., it has a relation which is neither empty nor does it contain all tuples over the domain of the database, then the set of all first-order sentences satisfied by B is PSPACE-complete [CM].

This happens to be quite a typical pattern. The expression complexity of the investigated languages is usually one exponential higher then the data complexity, and for both types of complexity we will show completeness in some complexity class. More specifically, we will show a hierarchy of languages whose data complexity is described by completeness in LOGSPACE, NLOGSPACE, PTIME, NPTIME and PSPACE, respectively, and whose expressions complexity is described by completeness in PSPACE, NPSPACE (=PSPACE), EXPTIME, NEXPTIME and EXPSPACE, respectively.

In this version of the paper proofs are only briefly sketched.

2. Databases, Queries and Complexity

We first recall some basic definition taken mainly from [CH2].

Definition: A *relational database* (or *database* for short) is a tuple $B=(D,R_1,\ldots,R_k)$ where $D\subseteq N$ is a finite set (N is the set of natural numbers) and for each $1\leq i\leq k$, $R_i\subseteq D^{a_i}$ for some $a_i>0$. The number a_i is called the *rank* of R_i and B is said to be of type $\bar{a}=(a_1,\ldots,a_k)$.

We will usually abbreviate the vector $R_1,\ldots,R_k$ by $\bar{R}$ and write $B=(D,\bar{R})$. Also, throughout the paper we will assume some standard encoding for databases. E.g., the database $B=(\{3,5,7\},\{\langle 3,5\rangle,\langle 5,7\rangle\})$ might be encoded by the string $(\{11,101,111\},\{\langle 11,101\rangle,\langle 101,111\rangle\})$.

Definition: A *query* of type $\bar{a}\rightarrow b$ is a partial function

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

such that if $B=(D,\bar{R})$ and $Q(B)$ is defined then $Q(B)\subseteq D^b$.

Chandra and Harel [CH1] define a query as *computable* if it is a partial recursive function and it satisfies what they call the "consistency criterion". The queries defined by the languages in this paper are all computable, so we will not address this point any further.

Definition: A *query language* (or *language* for short) is a set of expressions L and a meaning function μ such that for every expression e in L, $\mu(e)$ is a query. $C(L)$ is the class of queries defined by expressions in L, that is,

$$C(L)=\{Q \mid Q=\mu(e) \text{ for some } e\in L\}$$

We will write Q_e for $\mu(e)$ when μ is understood from the context.

Since queries are functions, we will measure their complexity by studying how difficult it is to recognize that a certain tuple belongs to the result of applying the query to the database. That is, we look at the problem as a recognition problem instead of looking at it as a computation problem. The result of applying the query to the database can be obtained by checking for all possible tuples (i.e., D^k) whether they belong to the result. For most languages dealt with in the paper, the complexity of computing the result is the same as the complexity of recognizing tuples in the result.

Definition: The *graph* of a query Q is the set

$$Gr(Q)=\{(\bar{d},B) \mid \bar{d}\in Q(B)\}.$$

The *graph* of a database B with respect to a language L is the set

$$Gr_L(B)=\{(\bar{d},e) \mid e\in L \text{ and } \bar{d}\in Q_e(B)\}.$$

The *data complexity* of a language L is the complexity of the sets $Gr(Q_e)$ for the expressions e in L. The *expression complexity* of a language L is the complexity of the sets $Gr_L(B)$.

Definition: A language L is *data-complete* (or D-*complete* for short) in a complexity class C if for every expression e in L, $Gr(Q_e)$ is in C, and there is an expression e_0 in L such that all sets in C are logspace reducible to $Gr(Q_{e_0})$. L is *expression-complete* (or E-complete for short) in a complexity class C if for every database B, $Gr_L(B)$ is in C, and there is a database B_0 such that all sets in C are logspace reducible to $Gr_L(B_0)$.

Definition: Let C be a complexity class. QC is the class of queries whose graph is in C, i.e.,

$$QC=\{Q \mid Gr(Q)\in C\}.$$

3. Logical Languages and their Complexity

The first query language to be studied here is the language of first-order logic, whose tuple oriented version is called *relational calculus* in [Codd].

Definition: Let L be the first-order language with equality, with no function symbols and with $R_1,R_2,\cdots$ as its predicate symbols. (Note: we will use R_i both as the formal symbol denoting a relation and as the relation itself; also the rank of R_i will be a_i). Let F be the language consisting of expressions of the form $\bar{x}\varphi(\bar{x})$ where φ is a formula of L and $\bar{x}$ is a vector of distinct variable containing all free variables of φ. If e is such an expression then $\mu(\bar{e})$ is a query Q_e of type $(a_1,\ldots,a_k)\rightarrow|\bar{x}|$ ($|\bar{x}|$ denotes the length of $\bar{x}$). Q_e is defined by

$$Q_e(D,\bar{R})=\{\bar{d}\in D^{|\bar{x}|} \mid \varphi(\bar{d}) \text{ is true in } (D,\bar{R})\}.$$

Let F^- be the quantifier-free version of F. That is, quantifiers are not allowed in expressions of F^-.

Clearly, F is more expressive then F^-.

Example: Let R_1 be a binary relation describing the flights of an airline company, i.e., $R_1(x,y)$ means that the company has a flight from city x to city y. The expression

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

represents the query Q of type $(2)\rightarrow 2$ which returns the composition of R_1 with itself, i.e., the pairs of cities that

are connected by exactly two flights.

A useful query is that which returns the pairs of cities that are connected by any positive number of flights. It was shown by Aho and Ullman [AU] that no expression in F represents this query. This is the motivation for enriching F by the transitive closure construct [Z1].

Definition: Let TF be the language obtained by adding to F expressions of the form $\tau.(x,y).\varphi(x,y)$, where $(x,y).\varphi(x,y)$ is an expression of F. If e is such an expression then $\mu(e)$ is a query Q_e of type $(\bar{a})\rightarrow 2$ defined by

$Q_e(D,\bar{R})=(Q_{(x,y),\varphi}(D,\bar{R}))^+$ ($^+$ stands for transitive closure).

Example: The desired query mentioned above is represented by $\tau.(x,y).R_1(x,y)$.

Let R_2 be a ternary relation describing flights of several airline companies, i.e., $R_2(x,y,z)$ means that company x has a flight from city y to city z. Consider the query which return all triples (u,v,w) such that v and w are connected by a positive number of flights all by the same company u. There is no way of representing this query in TF. This is the motivation for enriching F by the fixpoint construct.

Definition: Let YF be the language obtained by adding to F expressions of the form $\Upsilon R.\bar{x}.\varphi(\bar{x})$, where $\bar{x}.\varphi(\bar{x})$ is an expression of F and R is a predicate symbol of rank $|\bar{x}|$ that occurs positively in φ (i.e., each occurrence of R in φ is under an even number of negations). If e is such an expression then $\mu(e)$ is a query Q_e of type $(\bar{a})\rightarrow|\bar{x}|$. $Q_e(D,\bar{R})$ is a relation R such that $Q_{x,\varphi}(D,R,\bar{R})=R$ and for any relation R', if $Q_{x,\varphi}(D,R',\bar{R})=R'$ then $R\subseteq R'$. In other words, $Q_e(D,\bar{R})$ is the least fixpoint of the query $Q_{\bar{x},\varphi}$.

Chandra and Harel [CH2] have shown that all the expressions in YF define total queries.

Example: The above mentioned query is represented by the following expression of YF:

$$\Upsilon R.(x,y,z).(R_2(x,y,z)\vee\exists u(R_2(x,y,u)\wedge R(x,u,z))).$$

YF is more expressive then TF, because the query defined by the expression $\tau.(x,y).\varphi(x,y)$ of TF is also defined by the following expression of YF:

$$\Upsilon R.(x,y).(\varphi(x,y)\vee\exists z(\varphi(x,z)\wedge R(z,y))).$$

Consider now the query which returns the pair of cities connected by direct flights by an even number of airline companies. There is no way to express this query in YF [CH2]. It can however be expressed by using second-order existential quantification.

Definition: Let SF be the language obtained by adding to F expressions of the form $\bar{x}.\exists R.\varphi(\bar{x})$, where $\bar{x}.\varphi(\bar{x})$ is an expression of F. If e is such an expression and R is of arity a then $\mu(e)$ is a query Q_e of type $(\bar{a})\rightarrow|\bar{x}|$ defined by

$$Q_e(D,\bar{R})=\{\bar{d}\in D^{|\bar{x}|}\mid \text{there is a relation } R\subseteq D^a \text{ s.t.}$$

$$\varphi(\bar{d}) \text{ is true in } (D,R,\bar{R})\}.$$

Example: The above mentioned query is represented by the following expression of SF:

$$(u,v).\exists R.(\forall xyz((R(x,y)\wedge R(x,z))\rightarrow y=z)$$

$$\wedge(R(y,x)\wedge R(z,x))\rightarrow y=z)$$

$$\wedge(\neg R(x,y)\vee\neg R(z,x))$$

$$\wedge(R_2(x,u,v)\rightarrow\exists w(R(x,w)\vee R(w,x)))$$

$$\wedge(R(x,y)\rightarrow R_2(x,u,v)\wedge R_2(y,u,v)))).$$

It is easy to show that queries defined by expressions in YF can be defined by using second-order *universal* quantification. It is also true, though less trivial, that such queries can be defined by using second-order existential quantification. This can be shown by complexity

arguments [CH2] or by the technique of [JS]. Thus, SF is more expressive than YF.

The expressiveness relation between the languages defined so far can be summed up as

$$Q(F\ ) \subset Q(F) \subset Q(TF) \subset Q(YF) \subset Q(SF).$$

(We use $\subseteq$ to denote containment and $\subset$ to denote proper containment).

Let us now consider the data and expression complexity of F , F, TF, YF and SF, respectively. We use a fixed database $B_0=(D_0,R_0)$, where $D_0=\{0,1\}$ and $R_0=\{\langle 1\rangle\}$.

Theorem 1: F is D-complete and E-complete in LOGSPACE.

Proof: For LOGSPACE, hardness by a logspace reduction is trivial, so it suffices to show membership. For $e\in F^-$, $Gr(Q_e)$ is clearly in LOGSPACE (see next theorem). The fact that for every database B, $Gr_{Q_e}(B)\in LOGSPACE$ follows from the fact that Boolean expressions can be evaluated in logarithmic space [Ly]. □

Theorem 2: F is D-complete in LOGSPACE and E-complete in PSPACE.

Proof: To test whether $\bar{d}\in Q_{x,\varphi}(B)$, a straightforward algorithm cycles through all possible substitutions for the quantified variables. This algorithm has a logarithmic space data complexity and polynomial space expression complexity.

D-completeness: Trivial.

E-completeness: By reduction from Quantified Boolean Formulas of Stockmeyer [St] it follows that $Gr_F(B_0)$ is PSPACE-hard [CM]. □

The classification of F as having a logarithmic space data complexity is quite crude and can be further refined. For an expression $e\in F$, let $qn(e)$ be the number of quantifiers in e. Let $2DFA(k)$ denote the class of sets accepted by a two-way deterministic finite automaton with k heads. Recall that $LOGSPACE=\bigcup_{k\in N} 2DFA(k)$ [Ha].

Theorem 3: Let $e\in F$; then $Gr(Q_e)\in 2DFA(qn(e)+2)$.

Proof: $qn(e)$ heads are needed to move over the domain; thus giving an assignment of elements to the $qn(e)$ quantified variables, an additional head is needed to move over the assignment of elements to the free variables, and the last head is needed to move over the relations. The interaction between the last head and the first $qn(e)+1$ heads gives truth values to the atomic formulas in e. □

Interestingly, it is known that, for all $k\in N$, $2DFA(k)$ is properly contained $2DFA(k+2)$ [Ib], and it follows from the results of [CH2] that $C(\{e \mid e\in F \text{ and } qn(e)=k\})$ is properly contained in $C(\{e \mid e\in F \text{ and } qn(e)=k+2\})$.

Theorem 4: TF is D-complete in NLOGSPACE and E-complete in PSPACE.

Proof: To test whether $(a,b)\in Q_{\tau.(x,y).\varphi}(B)$ we guess a sequence $a=a_1,a_2,\ldots,a_k=b$ and test whether $(a_i,a_{i+1})\in Q_{(x,y),\varphi}(B)$ for $1\le i<k$. This algorithm has a nondeterministic logarithmic space data complexity, and a nondeterministic polynomial space expression complexity.

D-completeness: Let e_0 be $\tau.(x,y).R(x,y)$. $Gr(e_0)$ is NLOGSPACE-hard by reduction from Graph Accessibility problem [Jo].

E-completeness: Follows from the fact that PSPACE=NPSPACE and $F\subseteq TF$. □

Theorem 5: YF is D-complete in PTIME and E-complete in EXPTIME.

Proof: Let the fixpoint expression e be $\Upsilon R.\bar{x}.\varphi(\bar{x})$, where R is of rank $|\bar{x}|$. The algorithm of [CH2] for evaluating $Q_e(D,\bar{R})$ builds a sequence of increasing approximations for R starting with the empty relation. Since the cardinality of $Q_e(D,\bar{R})$ is bounded by $|D|^{|\bar{x}|}$, the length of the sequence has the same bound. Hence, this algorithm has a polynomial time data complexity and an exponential time expression complexity.

D-completeness: Follows by reduction from Path System

Accessibility problem [JL].

E-completeness: We show that any set accepted by an exponential time deterministic Turing machine is logspace reducible to $Gr_{YF}(B_0)$. The idea is to encode a computation of length 2^n by a $2n+m-$ary relation R (m depends on the machine in question).

$R(a_1,\ldots,a_n,a_{n+1},\ldots,a_{2n},a_{2n+1},\ldots,a_{2n+m})$ means that after i computation steps the machine has a symbol σ in its $j-$th tape square, where $(a_1,\ldots,a_n)$, $(a_{n+1},\ldots,a_{2n})$, and $(a_{2n+1},\ldots,a_{2n+m})$ are binary encodings of i, j and σ, respectively. The movements of the machine are simulated by the fixpoint operation. □

The algorithm outlined above for evaluating $Q_e(D,\bar{R})$ makes at most $|D|^{|x|}$ iterations, and the cost of each iteration is $O(||\varphi||\cdot|D|^{c||\varphi||})$ for some constant c. ($||\alpha||$ denotes the size of the encoding of α). The following theorem shows that the rank of R and the size of φ gives also a lower bound on the complexity of $Gr(Q_e)$.

Theorem 6: There is a polynomial p so that for every $k\in N$ there is an expression $e=\Upsilon R.\bar{x}.\varphi(\bar{x})$, with $|\bar{x}|=O(k)$ and $||\varphi||=O(p(k))$, such that $Gr(Q_e)\notin DTIME(n^{k-1})$.

Proof: There is a polynomial p so that given a deterministic Turing machine M with input alphabet Σ that operates in time $O(n^k)$, we can build an expression e with $|\bar{x}|=O(k)$ and $||\varphi||=O(p(||M||))$, such that, for every $s\in\Sigma^*$, one can construct in time $O(||s||)$ a database B and a vector $\bar{d}$ of size $O(||s||)$ such that $s\in L(M)$ iff $\bar{d}\in Q_e(B)$. The claim then follows from the fact that there is a Turing machine M with $||M||=O(k)$ that operates in time $O(n^k)$, but $L(M)\notin DTIME(n^{k-1})$ [HS]. □

Let $YF(k)$ be the class of expressions $\{e \mid e\in YF \text{ and } ||e||\leq k\}$. The theorem above shows that the hierarchy $\langle C(YF(k))\rangle_{k\in N}$ is infinite.

Theorem 7: SF is D-complete in NPTIME and E-complete in NEXPTIME.

Proof: To test whether $\bar{d}\in Q_{\bar{x}\exists R.\varphi}(D,\bar{R})$, a straightforward algorithm guesses a relation $R\subseteq D^a$ (a is the arity of R) and tests whether $\bar{d}\in Q_{x\varphi}(D,R,\bar{R})$. This algorithm has a nondeterministic polynomial time data complexity and a nondeterministic exponential time expression complexity.

D-completeness: Shown by reduction from the Clique problem of [Ka].

E-completeness: We show that any set accepted by a nondeterministic exponential time Turing machine is reducible to $Gr_{SF}(B_0)$. We use the encoding described in the proof of Theorem 5. □

Remark 1: Lower bounds for SF in the spirit of Theorem 6 can be shown in a similar manner.

Remark 2: Not only are YF and SF D-complete in PTIME and NPTIME, respectively, but also every query computable in nondeterministic polynomial time can be defined in SF, and if we assume that one of the relations of the database is a linear order on the elements of the domain then every query computable in polynomial time can be defined in YF^+, which is a slight extension of YF (more precisely, it allows existential quantification over conjunctions of equality formulas with expressions of YF). Fagin [Fa] has proven the first claim for Boolean queries, and by extending his argument we can show that $C(SF)=QNPTIME$. Studying the proof in detail, we can see that the second-order quantification is needed for two different purposes. First, it is used to define a linear order on the elements of the domain, and secondly it is used to say that there exists an accepting computation on the nondeterministic machine. If one of the relations in the database is already a linear order relation, and the machine is deterministic, then the computation can be simulated by the fixpoint operation. Thus, $C(YF^+,\leq)=QPTIME(\leq)$. (This fact has been also proven independently by Immerman [Im2]). Let $EVEN$ be the Boolean query that answer

positively for a database $(D,\bar{R})$ just in case that $|D|$ is even. Clearly, $EVEN \in QLOGSPACE$. However, it is known that $EVEN \notin C(YF^+)$ [CH2]. Consequently, $C(F) \subset QLOGSPACE$, $C(TF) \subset QNLOGSPACE$, and $C(YF^+) \subset QPTIME$. One may wonder whether $C(F,\leq) = QLOGSPACE(\leq)$ or whether $C(TF,\leq) = QNLOGSPACE(\leq)$. It follows from the results of [Im1] that the first statement does not hold, and we conjecture that neither does the second one.

4. Algebraic Languages and their Complexity

Our basic algebraic query language is based on the relational algebra of [Codd]. The language consists of the following:

Variables: $X_\emptyset^0, X_1^0, \ldots, X_\emptyset^1, X_1^1, \ldots, X_\emptyset^2, \ldots$

Terms: $D, R_i, X_i^a, \ldots$

and if t_1, t_2, t are terms then so are:

$t_1 \times t_2$	Cartesian product,
$t_1 \cup t_2$	union,
$\neg t$	complement,
$Proj_i(t)$	projection of column i,
$Perm_\theta(t)$	permutation by θ,
$Restrict_{i,j}(t)$	restriction of columns i and j.

(The numbers is the terms are assumed to be written in unary notation).

Definition: The language A consists of statements of the following form:

$X_i^a \leftarrow t$ where t is a term of rank a,

$(S_1;S_2)$ where S_1 and S_2 are statements.

The language A^- is the projection-free version of A. That is, projections are not allowed in terms of A^-.

The language BA consists of the statements of A and also

For $|t|$ *do* S , where t is a term and S is a statement.

The language LA consists of the statements of BA and also

While $t = \emptyset$ *do* S , where t is a term and S is a statement.

An expression in any of these languages is a pair (S,t) where S is a statement and t is an expression in the language.

The semantics of the language is as follows. Variable X_i^a has rank a. All variables are initialized to $\emptyset$. The values of terms D, R_i are from the database. The term $t_1 \times t_2$ has value $\{\langle \bar{d},\bar{e} \rangle \mid \bar{d} \in t_1, \bar{e} \in t_2\}$; $t_1 \cup t_2$ is set union if t_1 and t_2 have the same rank; $Proj_i(t)$ deletes the i-th column from t, having value $\{\langle d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_a \rangle \mid \bar{d} \in t\}$, where t has rank a and $i \leq a$; given a permutation θ on $\{1, \ldots, a\}$, where t has rank a, the value of $Perm_\theta(t)$ is $\{\langle d_{\theta(1)}, \ldots, d_{\theta(a)} \rangle \mid \bar{d} \in t\}$; and $Restrict_{i,j}(t)$ has the value $\{\bar{d} \mid \bar{d} \in t \text{ and } d_i = d_j\}$, where t has rank a, $i \leq a$, and $j \leq a$.

The semantics of "$\leftarrow$" and ";" is that of assignment and concatenation in the obvious way. *For* $|t|$ *do* S means: execute S $|t|$ times, where $|t|$ is determined upon entry to the loop. We call this *bounded looping*. *While* $t = \emptyset$ *do* S means: execute S as long as $t = \emptyset$, where t is evaluated dynamically. We call this *unbounded looping*.

An expression (S,t) defines a query $Q_{(S,t)}$ in the following manner: $Q_{(S,t)}(B)$ is the value of t after termination of S on B, or $\emptyset$ if S does not terminate on B.

It is easy to see that A is more expressive than A^-, BA is more expressive than A, and LA is at least as expressive as BA. However, it is not known whether LA is more expressive than BA. So we have

$$C(A^-) \subset C(A) \subset C(BA) \subseteq C(LA)$$

Theorem 8: A is D-complete in LOGSPACE and E-complete in PTIME.

Proof:

D-completeness: See next theorem.

E-completeness: Follows from the completeness in PTIME of the Circuit Value problem of [La]. □

Theorem 9: A is D-complete in LOGSPACE and E-complete in PSPACE.

Proof: For every $e_1 \in F$ there exists $e_2 \in A$ such that $Q_{e_1} = Q_{e_2}$, and for every $e_2 \in A$ there exists $e_1 \in F$ such that $Q_{e_1} = Q_{e_2}$ [CM,Codd]. Furthermore, the translation in both direction can be carried out in logarithmic space, and there is a constant c such that $||e_2|| \leq c||e_1||$ and $||e_1|| \leq c||e_2||$. (The linearity of the translation from A to F is not trivial!). Thus, the claim follows by Theorem 2. □

Theorem 10: BA is D-complete in PTIME and E-complete in EXPTIME.

Proof: Let $e = (S,t) \in BA$. The maximal rank of a term in e is bounded by $||e||$. Hence, the cardinality of all terms in e is bounded by $|D|^{||e||}$, and the number of steps in the execution is bounded by $|D|^{||e||^2}$. Thus, a straightforward evaluation has a polynomial time data complexity, and an exponential time expression complexity.

Chandra and Harel [CH2] have shown how to simulate a fixpoint query by unbounded looping. The same can be done with bounded looping. Furthermore, there is a constant c such that for every $e_1 \in YF$ there is $e_2 \in BA$ with $||e_2|| \leq c||e_1||$ and $Q_{e_1} = Q_{e_2}$, and the translation can be done in logarithmic space. Thus, the claim follows by Theorem 5. □

Theorem 11: LA is D-complete in PSPACE and E-complete in EXPSPACE.

Proof: Let $e = (S,t) \in LA$. As in the proof of Theorem 9, the cardinality of all terms in e is bounded by $|D|^{||e||}$. Thus, a straightforward evaluation has a polynomial space data complexity, and an exponential space expression complexity.

D-completeness: [CH2].

E-completeness: We show that any set accepted by an exponential space deterministic Turing machine is logspace reducible to $Gr_{LA}(B_0)$. The idea is to encode a configuration of length 2^n by a $2n+m$-ary relation R (m depends on the machine in question).

$R(a_1,..,a_n,a_{n+1},\ldots,a_{n+m})$ means that the tape has symbol σ in its j-th square, where $(a_1,\ldots,a_n)$ and $(a_{n+1},\ldots,a_{n+m})$ are binary encodings of j and σ, respectively. An unbounded loop simulates the unbounded computation of the machine starting from the initial configuration. □

Remark 3: Lower bounds for BA and LA in the spirit of Theorem 6 can be shown in a similar manner.

Let us now compare the expressiveness of the algebraic languages with that of the logical languages. We have already noted that $C(F) = C(A)$, and since projection corresponds to existential quantification, it follows that $C(F^-) = C(A^-)$. We also noted that $C(YF) \subseteq C(BA)$ and in fact also $C(YF^+) \subseteq C(BA)$. However, $EVEN \in C(BA)$ [Chan]. So $C(YF^+) \subset C(BA)$. The relationship between LA and SF is not clear. It is not known whether $C(SF) \subseteq C(LA)$, and it follows from the data complexity of LA and SF that $C(LA) \subseteq C(SF)$ if and only if PSPACE = NPTIME.

Remark 4: Since $C(YF^+) \subseteq C(BA)$ and $C(YF^+,\leq) = QPTIME(\leq)$, it follows that $C(BA,\leq) = QPTIME(\leq)$. In a similar manner it can be shown that $C(LA,\leq) = QPSPACE(\leq)$.

5. Concluding Remarks

The next table summarizes the results of the preceding sections.

Language	Data Complexity	Expression Complexity
F^-	LOGSPACE	LOGSPACE
F	LOGSPACE	PSPACE
TF	NLOGSPACE	(N)PSPACE
YF	PTIME	EXPTIME
SF	NPTIME	NEXPTIME
A^-	LOGSPACE	PTIME
A	LOGSPACE	PSPACE
BA	PTIME	EXPTIME
LA	PSPACE	EXPSPACE

We now sketch two extensions of the theory that will be described in detail in the full version of the paper.

Unlike the language F, which is closed under it logical connectives (the Boolean connectives and quantification), TF, YF, and SF are not closed. Closing this languages under their connectives gives us the languages T, Y, and S. For these languages we can define hierarchies of expressions and their queries (this is done in [CH2] for F and Y), and now we can investigate their expressiveness and complexity. In [CH2] it is shown that the first-order hierarchy is infinite and is E-complete in the polynomial hierarchy of [St]. In [Im2] it is shown that the fixpoint hierarchy collapses and $C(Y)=C(YF^+)$. From [St] it follows that the second-order hierarchy is D-complete in the polynomial hierarchy. In addition we can show that the transitive closure hierarchy is D-complete in the logarithmic hierarchy of [CKS], and that the second-order hierarchy is E-complete in an exponential hierarchy that can be define analogously.

We already noted that expression complexity measures the succinctness of the language in question. Thus, if we can somehow succeed in "squeezing down" our expressions, we would expect the expression complexity to increase, while the data complexity will not change. It is not clear whether it can be done for the logical languages, but it is easy to do that for the algebraic languages by using shorthands like exponentiation and binary notation. It turns out that this can add an exponential to the expression complexity. For example, The "squeezed" version of LA is E-complete in 2EXPSPACE (the class of language accepted by Turing machines operating in doubly exponential space).

Acknowledgement I am grateful to David Harel for arousing my interest in the subject of relational queries and for many stimulating discussions. I'd like also to thank Ashok Chandra for his insightful remarks, in particular for pointing out that changing the syntax of the language can change its expression complexity. Thanks are also due to Nick Pippenger for supplying me with useful references, and to Ron Fagin for commenting upon an earlier draft of the paper.

References

[AU] Aho, A.V., Ullman, J.D.: Universality of data retrieval languages. Proc. 6th ACM Symp. on POPL, 1979, pp. 110-117.

[Ba] Bancilhon, F.: On the completeness of query languages for relational databases. Proc. 7th Symp. on MFCS, 1978.

[CH1] Chandra, A.K., Harel, D.: Computable queries for relational databases. JCSS 21(1980), pp. 156-177.

[CH2] Chandra, A.K., Harel, D.: Structure and complexity of relational queries. Proc. 21st IEEE Symp. on FOCS, 1980, pp. 333-347. Also, to appear in JCSS.

[CH3] Chandra, A.K., Harel, D.: Horn clauses and the fixpoint query hierarchy. Proc. ACM Symp. on PODS, Los Angeles, March 1982.

[Chan] Chandra, A.K.: Programming primitives for database languages. Proc. 8th ACM Symp. on POPL, 1981, pp. 50-62.

[CKS] Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. JACM 28(1981), pp. 114-133.

[CM] Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational databases. Proc. 9th ACM STOC, 1977, pp. 77-90.

[Codd] Codd, E.F.: Relational completeness of database sublanguage. In *Data Base Systems* (Rustin, Ed.), Prentice Hall, 1972, pp. 65-98.

[Coop] Cooper, E.C.: On the expressive power of query languages for relational databases. Proc. ACM Symp. on PODS, Los Angeles, March 1982.

[Fa] Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of Computation* (R. Karp, ed), SIAM-AMS Proc. 7(1974), pp. 43-73.

[Ha] Hartmanis, J.: On non-determinacy in simple computing devices. Acta Info. 1(1972), pp. 28-36.

[HS] Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Trans. AMS 117(1965), pp. 285-306.

[Ib] Ibarra, O.H.: On two-way multihead automata. JCSS 7(1973), pp. 28-36.

[Im1] Immerman, N.: Number of quantifier is better than number of tape cells. JCSS 22(1981), pp. 384-406.

[Im2] Immerman, N.: Relational queries computable in polynomial time. This volume.

[Jo] Jones, N.D.: Space bounded reducibility among combinatorial problems. JCSS 11(1975), pp. 68-85.

[JL] Jones, N.D.: Laaser, W.T.: Complete problems in deterministic polynomial time. TCS 3(1977), pp. 105-117.

[JS] Jones, N.D., Selman, A.L.: Turing machines and the spectra of first-order sentences. J. of Symbolic Logic 39(1974), pp. 139-150.

[Ka] Karp, R.M.: Reducibility among combinatorial problems. In *Complexity of Computer Computation* (R.E. Miller and J.W. Thatcher, eds.), Plenum Press, 1972, pp. 85-103.

[La] Ladner, R.E.: The circuit value problem is logspace complete for P. SIGACT News 7(1975), pp. 18-20.

[Ly] Lynch, n.: Logspace recognition and translation of parenthesis languages. JACM 24(1977), pp. 583-590.

[Pa] Paredaens, J.: On the expressive power of the relational algebra. IPL 7(1978).

[St] Stockmeyer, L.J.: The polynomial time hierarchy. TCS 3(1977), p. 1-22.

[Zl] Zloof, M.: Query-by Example: Operations on the transitive closure. IBM Research Report RC5526, 1976.