WHAT REASONABLE FIRST-ORDER QUERIES ARE PERMITTED BY TRAKHTENBROT'S THEOREM?

Arnon Avron and Joram Hirschfeld

Raymond and Beverly Sackler
Faculty of Exact Sciences
School of Mathematical Sciences
Tel Aviv University
Ramat-Aviv, 69978 Israel

0. Introduction

Around 1950, B.A. Trakhtenbrot proved an important undecidability result (known, by a pure accident, as "Trakhtenbrot's theorem"): there is no algorithm to decide, given a first-order sentence, whether the sentence is satisfiable in some *finite* model. The result is in fact true even if we restrict ourselves to languages that has only one binary relation [Tra63]. It is hardly conceivable that at that time Prof. Trakhtenbrot expected his result to influence the development of the theory of relational databases query languages, but it did. This perhaps is not surprising in view of the following two facts:

1) The theory of relational databases is strongly rooted in logic and can easily be abstracted to make it a branch of mathematical logic.
2) The main interest in the theory of relational databases is in *finite* relations.

In the first section we explain those two points in more detail. Then, in section 2 of the present paper, we discuss the question: "What constitutes a reasonable query to a database?" The discussion naturally leads to a certain class of queries, known as the "domain independent" queries, as an ideal query language. At this point Trakhtenbrot's theorem appears as a major obstacle, since it easily implies that this ideal class is undecidable. *Real* query languages cannot be allowed to have this property. Section 3 outlines then several possible ways to circumvent this difficulty. The following section explains the

extra difficulties in applying them in case we allow function symbols in the language. The last two sections describes the two main approaches in greater depth, and explains how the difficulties with function symbols can be overcome (at least in principle). Section 5 is mainly based on [Hir91], while section 6 – on [AH91a].

1. The Logical Framework

A relational database is usually given in the form of a finite number of finite tables, each having a name. We start our discussion with a description of how such a database should be understood from an abstract logical point of view. As is usual in logical frameworks, we shall specify the languages used, their intended general semantics, and the corresponding notion of validity.

1.1 Languages. The languages used are standard first order languages with equality. The only restrictions are that the number of predicate and function symbols should be finite, while the set of constants should be effectively enumerable.

The demand for finiteness of the set of predicate symbols is due to the fact that all of them (except equality) are meant to correspond to tables in some specific database, and there are only finitely many such tables. The demand that there are only finitely many function symbols is less essential, but convenient. A given database can mention only finitely many such symbols, and the use of function symbols which are not mentioned in the database can be viewed as just an effective method of adding infinitely many new *constants* (this remark will become clearer after we describe the intended semantics). Finally, the demand concerning constants reflects a pragmatical aspect of the subject: tables and answers to queries should be effectively *listed*.

1.2 Theories. The theories which are of interest in the present context are finite sets of atomic *sentences* of the form $p(t_1, \ldots, t_n)$, where p is not the equality symbol. Such a theory D is just a first order representation of some database db, where for each p the set of sentences in D of the form $p(t_1, \ldots, t_n)$ is a full description of a corresponding table in db with the same name. We shall call such theories *db-theories*, and usually identify a database with the corresponding db-theory.

1.3 Semantics. The main factor that distinguishes *pure* relational database theory from other branches of mathematical logic is the structures which are taken to be of interest. These are *not* all the possible models of theories like above, but rather a restricted class of what might be called "preferred" (or admissible) models:

Definition. Given a language L and a *db*-theory D in L, an *L-db model* for it is a Herbrand structure for some effective extension of L with at most countable set of constants. This Herbrand structure should satisfy the following two conditions:

(i) Two terms of the domain are equal iff they are identical.

(ii) An atomic sentence (other than equality) is true in it iff it belongs to D. In particular: all the relations (except equality) should be *finite*.

Note. Given D in a language L, for every $0 \leq n \leq \infty$ there is up to isomorphism exactly one *L-db* model of D which has exactly n constants not belonging to L.

The fact that we confine ourselves only to Herbrand structures is a formal version of the demand that every element of a structure could be *named*, i.e.: corresponds to some syntactic term. This demand is a variant of what is known as the *domain closure assumption*. Condition (i) above means then that every such element has a *unique* name. This condition is known therefore as the *unique name assumption*. Finally, condition (ii) means that the given database is expected to have a full description of the relations. This is known as the *closed world assumption*.

There is one fundamental principle (already mentioned above) of relational database practice that dictates all these assumptions: we should be able to fully and unambiguously *list* everything which interests us. This applies, first of all, to the relations of D (i.e. the tables). It also applies to answers to queries, as we shall see below.

Note. In the above framework we ignore certain aspects of real database systems:

(1) In relational systems each column in a table has a name of its own, and the order of the columns is in principle irrelevant. Moreover: values which occur in different columns frequently have different types, and so the use of some many-sorted language might be more appropriate. These differences do not affect, however, the metatheory in an essential way (see, e.g., [Ull89]). The abstractions made here (and elsewhere)

are theoretically as harmless as the abstractions made in studying usual single-sorted first order languages.

(2) Much more important is the fact that in real databases some of the constants like "1" frequently have standard intended interpretation in a well known infinite domain (like N) and so commercial query languages allow the use of standard, *infinite* relations (like inequality between natural numbers) and even the use of standard functions, like $+$ and $\times$. An attempt to take this fact into account *would* change the theory completely. Here, however, we treat only what we call *pure* database languages, in which the only relations known and used are those which are explicitly represented as finite tables.

(3) The unique name assumption seems natural (and harmless) with respect to *simple* databases, in which we have no function symbols. It is somewhat restrictive when function symbols are allowed. It means, for example, that father_of (John) must be different from father_of (Jack), and that functions like $+$ and $\times$ should not be used in queries. In other words: function symbols are a way of naming and not of reflecting some truth about the universe. We believe that further research should be done concerning relaxing this assumption. This, again, will not be done here.

1.4 The Validity Concept. The relevant concept is not explicitly defined in the literature, but as in any application of a *logic* it[1] exists nevertheless and determines the way the logic is used. Thus the most common use of databases is in answering queries. Suppose now that D is a database. The simplest type of query to D is a "yes" or "no" question. What can be the form of such a query? The answer is: "Is a given sentence φ valid according to D"? The answer to this query will depend, of course, on the notion of validity we have in mind.

When we have semantics to some logic, the standard, obvious way to define validity in a theory D is as truth in every model of D. In our case we have the semantics of *db*-models. What is taken to be such a model depends, however, also on the *language* in which D is taken to be a theory (this is *not* uniquely determined!) Obviously, we should

[1] or a generalization: a consequence relation

use a language in which both φ (the query) and at least the relevant part of D can be expressed. The minimal such language is given in the following definition.

Definition.

(1) Given a database D and a formula φ, $L(D,\varphi)$ is the language in which:
   a) The predicate symbols are those which are mentioned in φ
   b) The constants and function symbols are those that are either mentioned in φ or in some sentence $p(t_1,\ldots,t_n)$ in D such that p is mentioned in φ.

(2) $L^n(D,\varphi)$, $0 \leq n \leq \infty$, is the language which is obtained from $L(D,\varphi)$ by adding n new constants[2] (Obviously, $L^0(D,\varphi) = L(D,\varphi)$.)

(3) $D|\varphi$ is the subset of D which consists of the sentences in D which belong to $L(D,\varphi)$.

(4) $\mathrm{Dom}_n(D,\varphi)$ is the unique model of $D|\varphi$ whose elements are the terms of $L^{(n)}(D,\varphi)$[3]

(5) $\mathrm{Dom}(D,\varphi)$ is $\mathrm{Dom}_0(D,\varphi)$ – the unique minimal admissible model of D.

It is obvious that every $L(D,\varphi)$-*db* model of $D|\varphi$ (see definition in 1.3) is isomorphic to $\mathrm{Dom}_n(D,\varphi)$ for some $0 \leq n \leq \infty$.

We are now ready to define validity:

Definition. Let D be a database and φ a sentence. $D \models_{db} \varphi$ iff φ is true in every $L(D,\varphi)$-*db* model of $D|\varphi$ (iff φ is true in $\mathrm{Dom}_n(D,\varphi)$ for every $0 \leq n \leq \infty$).

At this point it may seem natural to ask: why do we need to consider $\mathrm{Dom}_n(D,\varphi)$ for $n > 0$? After all, all the models other than $\mathrm{Dom}(D,\varphi)$ contain elements which are neither referred to in φ nor in the relevant fragment $D|\varphi$ of D. Why should they be important? The answer is that frequently we do need to know the truth value of φ in a model other than $\mathrm{Dom}(D,\varphi)$. For example, in order to evaluate the truth value of $\varphi \vee \psi$ in (say) $\mathrm{Dom}(D,\varphi \vee \psi)$, we have to evaluate the truth value of φ in $\mathrm{Dom}(D,\varphi \vee \psi)$. But $\mathrm{Dom}(D,\varphi \vee \psi)$ may be isomorphic to $\mathrm{Dom}_n(D,\varphi)$ for some $n > 0$!

We can look at the issue also from a little bit different (although strongly related)

[2] There are several methods of determining $L^{(n)}(D,\varphi)$ exactly, but we shall not bother to do so. It is obvious that the precise identity of the new constants is not important.

[3] $\mathrm{Dom}_n(D,\varphi)$ $n > 0$ is unique up to isomorphism – see previous footnote.

point of view. There are certain basic properties that we would like our validity notion to have, but obtain only if we consider $\mathrm{Dom}_n(D,\varphi)$ for all n. For example:

- If either φ or ψ is valid then so is $\varphi \vee \psi$
- If both φ and ψ are valid then so is $\varphi \wedge \psi$.[4]

A third reason that we would like to mention is pragmatical. As we noted above, in practice the constants we use often have a fixed intended meaning in an apriorily given structure that might even be infinite. We *are* aware then of the existence of elements in the domain of discourse which do not belong to $\mathrm{Dom}(D,\varphi)$. There might be cases in which these extra elements might be relevant, and we don't want our theory to exclude this possibility (this explains why we allow the case $n=\infty$ in the definition of $\mathrm{Dom}_n(D,\varphi)$).

2. What Queries Are "Reasonable"?

The notion of a query to a relational database generalizes in many ways that of a system of equations. Such a system is usually just a conjunction φ of atomic formulae of the form $t_1 = t_2$ (or, rarely, a disjunction of such conjunctions). The (free) variables $x_1,\ldots,x_n$ of φ are the "unknowns" and a "solution" is a set of irreducible closed terms $s_1,\ldots,s_n$ (like "3" or "2π") such that $\varphi(s_1/x_1,\ldots,s_n/x_n)$ is valid. In relational database theory, in contrast, *any* formula $\varphi(x_1,\ldots,x_n)$ is a potential "query". Given a database D a solution is then any tuple $(s_1,\ldots,s_n)$ of closed terms such that $D \models_{db} \varphi(s_1/x_1,\ldots,s_n/x_n)$ (note that because of the unique name assumption, every closed term is "irreducible").

When we learn in school the subject of systems of equations, all the efforts are almost entirely concentrated on what might be called "reasonable" systems. These are systems the solutions of which can be *listed*. This means, first of all, that the set of solutions is finite, and also that the elements of this set can all effectively be found.[5] An answer to such a "reasonable" system of equations φ is then just a formula ψ of the form $(x_1 = t_1^1 \wedge \cdots \wedge x_n = t_n^1) \vee \cdots \vee (x_1 = t_1^k \wedge \cdots \wedge x_n = t_n^k)$ (where t_i^j are closed, irreducible terms

[4] The converse of this fails, however. Thus $(\exists x\ x \neq b) \wedge a = a$ is valid, while $\exists x\ x \neq b$ is not. This anomaly can be explained in several ways, but we shall not discuss this issue here.

[5] Trigonometric equations are an interesting sort of exception, but here too "reasonable" equations are such that the solutions in any given finite interval can be listed.

and $x_1, \ldots, x_n$ are the "unknowns") such that $\forall x_1, \ldots, x_n \ \varphi \leftrightarrow \psi$ is valid.[6] Now exactly the same criterion for "reasonableness" applies to queries in relational databases, and the form of an answer to a reasonable query is expected to be just the same (A "table" is no more than a compact way of representing a formula ψ as above). Only the notion of validity which is used for defining an answer ψ to a query φ (given a database D) is, of course, $D \models_{db}$. One should note that according to this definition, the property of being a "reasonable" query is *relative* to a given database D. A query can be reasonable relative to one database but unreasonable relative to another.

The condition of "reasonableness" (relative to D) that has just been described can very naturally be split into the following two conditions.

(I) "Finite" Relative Domain Independence ([Ki88]): A formula φ is (finitely) domain independent (d.i.) relative to D if the same set of tuples of closed terms satisfies it in all (finitely generated) admissible models of $D|\varphi$ (i.e. $\mathrm{Dom}_n(D, \varphi)$ $0 \leq n \leq \infty$, or just finite n in the case of finite relative domain independence).

(II) Relative Safety ([AGS86], [Ki88]): A formula φ is safe relative to D iff for every n it has a finite set of solutions in $\mathrm{Dom}_n(D, \varphi)$.

Notes. 1) The two notions of relative d.i. are in fact equivalent. This was shown in case there are no function symbols in [Ki88] and in the general case in [AH91a], (see section 6 below).

2) There are other notions of "safety" which are used in the literature – see [Ki88] for a survey. The one above is what Kiffer uses in [Ki88].

Obviously, in case there are no function symbols a query is reasonable (relative to D) iff it is both d.i. and safe (relative to D). It is also clear that under the same assumption $\mathrm{Dom}_n(D, \varphi)$ is finite for $n < \infty$, and so relative d.i. implies relative safety. It was conjectured in [Ki88] and proved in [AH91a] that these facts hold also in case we do have function symbols (see section 6). It follows that relative reasonableness and relative d.i. are two equivalent notions. It is no wonder, therefore, that domain independence is

[6] It is worth noting that in trigonometric equations ψ has the form of a disjunction of certain simple *existential* formulae of the form $\exists k(x = t(k))$, where k is the only free variable of t.

widely taken as the crucial property that characterizes "reasonable" queries.

The notion of d.i. and of safety that we have introduced are *relative* notions. A query φ can, for example, be d.i. relative to D_1 but not domain dependent relative to D_2. One of the most important features of *real* databases, however, is that they change all the time through updating. Hence a query which is reasonable at a certain moment might become unreasonable a few moments later. It is very desirable to avoid such a situation. A really "good" query is one that is *always* reasonable, i.e.: domain independent relative to every D.[7] Formulae which have this property are called *universally domain independent*, or just *domain independent* (see, e.g. [Ull89]).

The upshot of this discussion is the widely accepted view, that the class of universally d.i. formulae forms the ideal query language. (see, e.g., [To87], [Ull89]). As many other ideals, however, this ideal also cannot be fully achieved. It was proved in [DiP69] that this class is undecidable even in the simple case of languages with no function symbols. It follows that it cannot be identical with the set of queries of any concrete, effective query language.

It is in the proof of this undecidability result that Trakhtenbrot's theorem plays a decisive role. To see how, take any sentence A in a language with no function symbols. Define $\varphi(x) = \neg A \wedge x = x$. It is not difficult to show that φ is d.i. iff A is true in all finite structures. It follows that had the notion of d.i. been decidable, so would have been the notion of validity in all finite structures. But Trakhtenbrot's theorem says that this is not the case!

3. Three Approaches to How to Be As Reasonable As Possible

3.1. Concentrating On One Domain. This approach requires that together with a query the user will specify which domains (among, e.g., the $\mathrm{Dom}_n(D, \varphi)$) he has in mind. The trouble is that often (most of the time?) the user does not know this, or might need some extra knowledge to determine it. This extra knowledge might refer to possible changes of the database in the future.

[7] This corresponds to a stronger notion of validity: φ is strongly valid if $D \models_{db} \varphi$ for every D.

Another version of this approach is to always take $\mathrm{Dom}(D, \varphi)$ as the intended domain. We have seen already the shortcomings of this approach.

3.2. Using Approximations of the Ideal. This is the approach taken by all present query language. The idea is described in [Ull89, p.151] as follows: "We shall first consider what we really would like concerning relational calculus expressions. This property, called "domain independence", is a semantic notion; it will be seen that it is impossible to tell, given a formula, whether the formula satisfies the property. After defining "domain independence" we shall therefore look for approximations to the ideal".

We now give what seems to us a proper definition of a "proper approximation":

Definition. A class L of formulae is a proper query language if:

(i) L is decidable

(ii) Every formula in L is domain independent

(iii) There is a procedure that given some domain independent query φ produces a query φ' in L such that φ and φ' are equivalent.

Note. By (b) and (c) every two proper query languages are equivalent: they have the same expressive power as the class of domain independent queries.

The question of the existence of proper query languages and methods for obtaining them are discussed in section 5.

3.3 Using Relative Domain Independent Queries. The proper languages form an attempt to overcome the problem of the undecidability of the class of universal domain independent queries by *shrinking* it to a smaller class, without losing expressive power. Here, in contrast, the approach is to extend in each case this class to some subclass of the formulas which are d.i. relative to the given database D. Since the identity of this subclass might depend on D, this approach requires the use of an algorithm that given D and φ determines whether φ belongs to the subclass associated with D. As it happens, one can in principle use, for any D, simply the whole class of formulas which are d.i. relative to D. More on this – in section 6 below.

4. The Problems With Languages With Function Symbols

The choice of (relative) d.i. as the crucial property was due, we recall, to the fact that (relative) d.i. entails (relative) safety. This, however, was obvious only for *simple* languages (i.e.: languages with no function symbols), where $\mathrm{Dom}(D, \varphi)$ is finite. It was not at all obvious for *generalized* languages (i.e.: languages *with* function symbols). Still, as noted above, this turned out to be the case. There is, however, another problem which was hidden in the simple case. A query is intuitively reasonable, we recall, if we can *list* all the solutions. This means not only that they are finite in number, but also that we can effectively find them all. This is easy if φ is relatively d.i. and $\mathrm{Dom}(D, \varphi)$ is finite. But if there are function symbols then $\mathrm{Dom}(D, \varphi)$ is infinite, and we might have a problem even if there are only finitely many solutions. In [Ki88] another property, which a "reasonable" query language should have, is therefore introduced. We split it here into two properties:

Definition. (1) A class Q of formulae is strongly capturable {relative to a database D} if each $\varphi \in Q$ is domain independent {relative to D} and there is an algorithm that given $\varphi \in Q$ and D computes the answer to φ.

(2) A class Q of formulae is weakly capturable relative to D if there is an algorithm that given $\varphi \in Q$: (i) determines if it is d.i. relative to D and (ii) Computes the answer if it is.

Obviously, for simple languages the class of formulae which are d.i. relative to D is strongly capturable relative to D – simply compute the answer in $\mathrm{Dom}(D, \varphi)$. Again this is far from being obvious in the generalized case. Moreover, it was shown in [AGS86] (see also [Ki88]) that for simple languages relative d.i. and relative safety are both decidable. It follows immediately that for such languages the whole class of first order queries is weakly capturable ([Ki88]). For generalized languages, on the other hand, decidability of relative d.i. does not immediately guarantee this last result. Still, it was proved in [AH91a] that all the results mentioned above concerning the simple case hold also for the generalized one, although the proofs are much more difficult. An outline of them is given in section 6.

Turning now to the approach of "approximating the ideal", we note that many different query languages of the type we called "proper" (see 3.2) have been suggested in

the literature: [Co72] ("range separable formulae"), [Ni82] ("range restricted formulas"), [De82] ("evaluable formulas"), [To87] ("allowed formulas") and [Ull89] ("Syntactically safe formulae"). Topor (in [To87]) was the first to justify his class from a point of view which is similar to the one presented here (by showing that every d.i. query is equivalent to an "allowed" query). However, all these classes (including Topor's) were suggested for simple databases, and Topor admits that there are serious difficulties in finding a good class of queries in case there are function symbols (see [To87], [TS88]). Such a class of queries (the *secured* queries) was first introduced and shown to be adequate in [Hir91]. In section 5 we review Topor's and Hirschfeld's results.

The following lemma is the key to the difference in the present context between the simple and the generalized case.

Lemma. *1) If φ does not mention function symbols then there is a formula $D_\varphi(x)$ which for every simple database D describes the subdomain* $\mathrm{Dom}(D,\varphi)$ *inside every other domain M (note that D_φ depends only on φ, not on D).*

2) If D or φ mention at least one function symbol then there is no formula that defines $\mathrm{Dom}(D,\varphi)$ *inside (say)* $\mathrm{Dom}_1(D,\varphi)$.

Proof of 1. The formula D_φ is a disjunction of formulae of two kinds:

(i) $\exists y_1 \cdots \exists y_{i-1} \exists y_{i+1} \cdots \exists y_{n_j} R_j(y_1, \ldots, y_{i-1}, x, y_{i+1}, \ldots, y_{n_j})$, where R_j ranges over the relations mentioned in φ, n_j is the arity of R_j and $1 \le i \le n_j$.

(ii) $x = c$, where c is a constant which is mentioned in φ.

The second part follows from one of the lemmas (number 5) which are used in [Hir91] for the proof of the theorem in section 5. We only note that if either D or φ contains a function symbol then $\mathrm{Dom}(D,\varphi)$ is infinite and the formula $D_\varphi(x)$ of part (i) describes only a finite subset of it.

5. Proper Query Languages

As we noted at the end of the last section, many suggestions for what we call "proper query languages" were suggested in the case of simple language. We now describe two examples, both from [To87], which represent two different approaches to the problem.

Example 1. The class of allowed formulae

Definition. A variable x is *pos* (positive) in a formula W if one of the following cases holds.

- x is pos in $p(t_1, \ldots, t_n)$ if x occurs in $p(t_1, \ldots, t_n)$ and p is not $=$;
- x is pos in $x = c$ or $c = x$ if c is a constant;
- x is pos in $\neg F$ if x is neg in F;
- x is pos in $F \wedge G$ if x is pos in F or x is pos in G;
- x is pos in $F \vee G$ if x is pos in F and x is pos in G;
- x is pos in $F \rightarrow G$ if x is neg in F and x is pos in G;
- x is pos in $\exists y F$ if x is pos in F.

Similarly, x is neg (negative) in W if one of the following cases holds.

- x is neg in $\neg F$ if x is pos in F;
- x is neg in $F \wedge G$ if x is neg in F and x is neg in G;
- x is neg in $F \vee G$ if x is neg in F or x is neg in G;
- x is neg in $F \rightarrow G$ if x is pos in F or x is neg in G;
- x is neg in $\forall y F$ if x is neg in F.

Definition. A formula W *is allowed* if the following conditions hold:

(1) every free variable in W is pos in W;

(2) for every subformula $\exists x F$ of W, x is pos in F;

(3) for every subformula $\forall x F$ of W, x is neg in F.

This is a typical bottom-up construction, which assures that when W is evaluated all the variables range over subsets of $\mathrm{Dom}(D, W)$ which do not depend on the actual domain in which W is evaluated. This follows from the fact that inside the building blocks of W (the atoms) the variables satisfy some conditions (of positiveness and negativeness) so that when the formula is inductively put together these conditions are also put together to produce the desired d.i. property. The bottom-up construction means that a recursive test is needed to check whether a given query is allowed.

In [To87] it is shown that the class of allowed formulae is indeed proper in our sense.

Example 2. Relativized Queries. For each formula φ let φ' be the formula obtained from φ by relativizing all the quantifiers and free variables of φ to D_φ of section 4. (this means that we obtain first φ'' from φ by inductively replacing each quantification $\exists x\psi$ by $\exists x(D_\varphi(x) \wedge \psi'')$ and $\forall x\psi$ by $\forall x(D_\varphi(x) \rightarrow \psi'')$. Then if $x_1, \ldots, x_n$ are the free variables of φ, we let φ' be $\varphi'' \wedge D_\varphi(x_1) \wedge \cdots \wedge D_\varphi(x_n))$. *A relativized query* is a query of the form φ' (for some φ).

Now assume that $L(D, \varphi)$ is simple. It is easily shown then by induction that for every $a_1, \ldots, a_n \in \mathrm{Dom}_k(D, \varphi)$; $\varphi'(a_1, \ldots, a_n)$ is true in $\mathrm{Dom}_k(D, \varphi)$ iff $a_1, \ldots, a_n \in \mathrm{Dom}(d, \varphi)$ and $\varphi(a_1, \ldots, a_n)$ is true in $\mathrm{Dom}(D, \varphi)$. It follows that φ' is always d.i. and that if φ is also d.i. then φ is equivalent to φ' in all admissible domains. It is also very easy to determine if a given formula is a relativized one. It follows that the class of relativized queries is a proper query language.

The relativization approach has the advantage of securing a query top-down. We may start with any formula and add the requirement that all its variables should range over the subset described by the formula D_φ. An obvious possible generalization is the following: start by defining a collection of "simple" formulae that define "absolute permissible sets" and then allow any formula in which it is specified over what absolute set each variable should range. In a way the "simple" formulae act like types, and a query is any formula whose variables are typed. This "top-down" nature can be made explicit if we allow exterior variables which are already typed to serve as parameters in the typing of the inner variables. For example, assume that $A(x, y)$ is any atomic formula and that $\psi(x)$ is $\exists y A(x, y)$. Assume also that $p(x)$ and $R(x, y)$ are relations in the database. Then $x = y \vee R(x, y)$ is not a good restriction, since it is not d.i., but once x is restricted to $P(x)$ it becomes a good restriction for y. The restricted formula will be then: $P(x) \wedge \exists y[(R(x, y) \vee x = y) \wedge A(x, y)]$. The user friendliness of this approach becomes clear if we separate the query itself (which may be any formula) from the typing of the variables. The above query will then look like this:

$$\exists y A(x, y)\{x : P(x),\ y : R(x, y) \vee x = y\} \ .$$

While this idea is of interest already for simple languages, it becomes essential when we turn to generalized ones, since there no uniform tool like D_φ is available (see the end

of section 4).

In the following definition the *absolute* formulas are the "simple" formulae over which the quantifiers can range, *bounded* formulas are formulas in which the quantified variables are restricted by absolute formulas, and *secured* formulas are formulas in which all the variables are restricted by absolute formulas. The class of secured formulas is the one which was suggested in [Hir91] as a proper approximation in the generalized case.

Definition.

1) Absolute Formulas.

a) Every atomic relation $R\,(t_1(\overline{x}), \cdots, t_k(\overline{x}))$ is absolute, while $x = t$ is absolute iff t is variable free.

b) If A and B are absolute then so is $A \wedge B$. If in addition they have exactly the same free variables then also $A \vee B$ is absolute. (But not for example $R(x) \vee S(y)$).

c) If A is absolute then so is $\exists y A$.

d) (Closure under the application of definable functions:) If $\varphi(x_1, \ldots, x_n)$ is absolute with $x_1, \ldots, x_n$ actually free in φ and if $t_1, \ldots, t_k$ are terms whose variables are among $x_1, \ldots, x_n$ then the following formula is absolute:

$$\psi(y_1, \ldots, y_k) \equiv \exists x_1 \cdots \exists x_n \left[\varphi(x_1, \ldots, x_n) \wedge y_1 = t_1 \wedge \cdots \wedge y_k = t_k\right] \ .$$

2) Bounded Formulas.

a) Every quantifier free formula is bounded.

b) If φ is bounded and if $A(x)$ is absolute with x actually free in $A(x)$ then the following are bounded:

(i) $\exists x(A(x) \wedge \varphi)$

(ii) $\forall x(A(x) \rightarrow \varphi)$

3) Secured Formulas. If $\varphi(x_1, \ldots, x_n)$ is bounded and if $A(x_1, \ldots, x_n)$ is absolute with $x_1, \ldots, x_n$ the free variables of A then $A \wedge \varphi$ is secured.

Remark. The definitions can be modified to allow more variables as parameters, provided we keep trace of them and they will be taken care later. In particular an equation $y = t(x_1, \ldots, x_n)$ is absolute modulo any choice of n out of the $n+1$ variables occurring. This will yield a somewhat more flexible language of secured queries. We do not need it at the moment since the definition above suffices to give us a proper language.

Clearly the language of secured queries is decidable. It follows therefore from the following theorem that this language is a proper query language for first order relational databases with functions.

Theorem.

a) Every secured query is domain independent.

b) Every domain independent query is equivalent to some secured query.

The proof of (a) is simple: by easy induction it is seen that D and every absolute formula defines in every domain for D and φ the same subset. It is then shown that if $\psi(x_1, \ldots, x_n)$ is bounded then for every $t_1, \ldots, t_n$ (terms without variables) in the language, $\psi(t_1, \ldots, t_n)$ has the same truth value in all the domains for D and φ that include $t_1, \ldots, t_n$. From this it follows that if $\varphi(\overline{x})$ is secured then all the solutions anywhere are already in $\mathrm{Dom}(D, \varphi)$ and they do not change from one domain to another.

The proof of (b), on the other hand, is quite complicated, and we shall not include it here. The reader can find it in [Hir91].

6. Using the Relative Notions

We start again with the simple case:

Lemma ([AGS86],[Ki88]). *Let D be a database and let $\varphi(x_1, \ldots, x_k)$ be a formula whose number of quantifiers and free variables is (together) n. Let M_1 and M_2 be admissible domains for D and φ that have at least n elements outside $\mathrm{Dom}(D, \varphi)$. Let $a_1, \ldots, a_k$ in M_1 and $b_1, \ldots, b_k$ in M_2 have the following properties:*

(i) For $i = 1, \ldots, k$, if $a_i \in \mathrm{Dom}(D, \varphi)$ or $b_i \in \mathrm{Dom}(D, \varphi)$ then $a_i = b_i$

(ii) $a_i = a_j$ iff $b_i = b_j$.

Then $\varphi(a_1, \ldots, a_k)$ is true in M_1 iff $\varphi(b_j, \ldots, b_k)$ is true in M_2.

The proof is by an easy structural induction. Conditions (i) and (ii) take care of the atomic case, while the assumption on the size of M_1 and M_2 is used in the case of $\exists x\psi$.

Note. It is not difficult to see that this lemma implies that if φ has the same solutions in all the finite models for D then it has the same solutions also in the infinite domain. Hence finite (relative) d.i. implies (relative) d.i.

Theorem. *In a simple language and databases, the class of the first order formulas is weakly capturable and that of relative d.i. – strongly capturable.*

Proof: For the first part, let φ be a simple formula with n_0 quantifiers and free variables. We first check if it has in $\mathrm{Dom}_{n_0}(D, \varphi)$ some solution which does not entirely lie in $\mathrm{Dom}(D, \varphi)$. If it does then by the last lemma, φ has new solutions in $\mathrm{Dom}_{n_0+1}(D, \varphi)$ and so is not d.i. relative to D. If, on the other hand, all the solutions in $\mathrm{Dom}_{n_0}(D, \varphi)$ are inside $\mathrm{Dom}(D, \varphi)$, then by the same lemma φ has the same set of solutions in all the domains $\mathrm{Dom}_n(D, \varphi)$ with $n \geq n_0$. This gives us the following constructive criterion for checking d.i. relative to D: φ is d.i. relative to D iff (a) All the solutions in $\mathrm{Dom}_{n_0}(D, \varphi)$ are inside $\mathrm{Dom}(D, \varphi)$, and (b) φ has the same set of solutions in all the domains $\mathrm{Dom}_n(D, \varphi)$ with $0 \leq n \leq n_0$ (note that these are all finite).

The second part easily follows from the first.

We now turn to generalized languages, and outline the treatment that we gave them in [AH91a]. Full proofs can be found in that paper (see also [AH91b]).[8] The main idea is a reduction of a query to several easy-to-handle normal forms, one of them uniform for sufficiently large domains.

Definition.

(1) *A special primitive formula* (Spf) is a formula of the form:

$$\phi(\vec{x}, \vec{y}) = \exists \vec{v}\, [P(\vec{x}, \vec{y}, \vec{v}) \wedge N(\vec{y}, \vec{v})]$$

where:

[8] Certain resemblance between our procedures and the work of Malcév in [Mal71] should be noted here.

(i) P is of the form $\wedge x_i = t_i(\vec{y}, \vec{v})$.

(ii) N is a conjunction of inequalities of the form variable $\neq$ term.

(iii) Each x_i occurs exactly once in ϕ.

(iv) Each v_i occurs at least once in P.

(2) *A special existential formula* is a disjunction of Spfs.

Theorem.

a) *Given D, φ and n one can construct a special existential formula $E_n^{D,\varphi}$ which is equivalent to φ in $\mathrm{Dom}_n(D, \varphi)$.*

b) *φ has a finite number of solutions in $\mathrm{Dom}_n(D, \varphi)$ iff each disjunct of $E_n^{D,\varphi}$ is of the form*

$$x_1 = t_1 \wedge x_2 = t_2 \wedge \cdots \wedge x_k = t_k$$

where $x_1, \ldots, x_k$ are the free variables of φ and $t_1, \ldots, t_k$ are closed.

c) *If $E_n^{D,\varphi}$ does not have this form then no proper subdomain of $\mathrm{Dom}_n(D, \varphi)$ includes all the solutions (in $\mathrm{Dom}_n(D, \varphi)$).*

Corollary. *Relative d.i. $\Longrightarrow$ relative safety.*

The formula $E_n^{D,\varphi}$ depends on n. In order to get a more uniform normal form, we add to the language a new predicate, $G(x)$, with the intended meaning: "x is a generator" (i.e.: a simple constant term). It is worth noting that $G(x)$ is definable in the language (since we assume only a finite number of function symbols!). But as is often the case with quantifiers elimination, we need this predicate to be atomic.

Theorem. *Given D and φ, one can constructively find a number $n_{D,\varphi}$ and a formula $E^{D,\varphi}$ (in the extended language) which has the same properties as $E_n^{D,\varphi}$ (i.e. (a) – (c) of the previous theorem) for $n_{D,\varphi} \leq n \leq \infty$, except that the $N(\vec{y}, \vec{v})$ part may contain formulae of the forms $\pm G(y_i), \pm G(v_i)$.*

Corollaries.

1) *If φ is safe relative to D then it has the same set of solutions in all $\mathrm{Dom}_n(D, \varphi)$ s.t. $n_{D\varphi} \leq n \leq \infty$.*

2) *φ is safe relative to D iff it has a finite set of solutions for all* $\mathrm{Dom}_n(D,\varphi)$ *s.t.* $0 \leq n < n_{D,\varphi}$ *and in* $\mathrm{Dom}_\infty(D,\varphi)$*, and these are all decidable questions.*

3) *φ is d.i. relative to D iff it is safe relative to D and has the same set of solutions for all n s.t. $0 \leq n \leq n_{D,\varphi}$.*

4) **(i)** *The class of first order queries is weakly capturable.*

(ii) The class of relatively safe queries is strongly capturable.

Moreover.

5) It is decidable whether a query has a finite number of solutions in all finitely generated domains (over a given D).

6) A formula is domain independent relative to D iff it has the same set of solutions in all finitely generated domains (over D).

The proof of these corollaries from the last two theorems is similar to the proof of the analogous results in the case with no function symbols.

We started this paper with a remark about Trakhtenbrot's theorem, and it is appropriate to conclude it with another remark concerning it.

Recall that DiPaola used Trakhtenbrot's theorem to show that there is no algorithm to determine if a given formula is universally domain independent. Above we have shown, in contrast, that there *is* an algorithm to determine whether a given formula is d.i. relative to a given database. This might hint that there is a relativized version of Trakhtenbrot theorem that does *not* hold. This indeed is the case, and though it is not very deep, it is worth mentioning. Given a finite structure M in some simple (i.e., functionless) language, let its *actual diagram* be the set of elements which satisfy (as part of a tuple) at least one of the atomic relations. Then:

Theorem 6. *There is an algorithm which given a statement φ in a simple first order calculus and a number n_0 determines if φ has a (finite) model with actual diagram of size* $\leq n_0$.

Thus it is true that the smallest model for φ may be unpredictably large (Trakhtenbrot), but it may only happen if the actual diagram becomes also impredictably large.

The proof follows from the last lemma above. For a fixed n_0 there are only finitely many actual diagrams $D_1, \ldots, D_\ell$ and we need only check φ in $\mathrm{Dom}_n(D_i, \varphi)$ with $i \leq \ell$ and $n \leq n_\varphi$.

7. Conclusion

We have seen that there are two possible approaches to the problem of making "reasonable" first-order queries to a pure relational database:

1) If we want to use only queries that are reasonable with respect to any possible database then we should use some syntactically defined, decidable approximation of the ideal language (that of universally domain-independent queries). There are several choices available here, but in a certain precise sense they are all equivalent.
2) If we are only interested in making reasonable queries to the particular database we have at a given moment then any first order formula might be a candidate. We have, however, to apply first a special algorithm which checks if the formula is indeed reasonable relative to the given database. We have seen that such an algorithm exists. In order to make this approach practical a more efficient one than that presented here should be found.

We have seen also that despite the apparent difficulties, as long as all the function symbols are interpreted as generative, there is no significant *theoretical* difference between the simple case in which we do not have function symbols and the generalized one in which we do.

All the results above were based on the assumption that we are dealing only with *pure* query languages, in which the only relation symbols (except equality) are the names of the tables in the database. We believe that in order to make the theory more applicable, it is important to investigate impure languages as well. In particular: languages with symbols for standard infinite relations like "$<$". The case in which not all function symbols are generative should also seriously be investigated.

References

[AGS86] A.K. Aylamazyan, M.M. Gilula, A.P. Stolbashkin, G.F. Schwartz, *Reduction of the relational model with infinite domain in the case of finite domains*, Proc. USSR acad. of Science (Doklady) 286(2): 308-311 (1986).

[AH91a] A. Avron and J. Hirschfeld, *Queries Evaluation, Relative Safety, and Domain Independence in First Order Databases with Functions*. To appear in the Journal of Methods of Logic in Computer Science.

[AH91b] A. Avron and J. Hirschfeld, *On First-order Database Query Languages*, Proc. 6th IEEE Symp. Logic in Computer Science, Amsterdam, July 1991, pp. 226-231.

[Co72] E.F. Codd, *Relational Completeness of Database Sublanguages*, in **Database Systems** (R. Rustin, Ed.), Prentice-Hall (1972), pp. 65-98.

[De82] R. Demolombe, *Syntactical characterization of a subset of domain indpendent formulas*. Technical report, ONERA-CERT, Toulouse (1982).

[DiP69] R.A. Di Paola, *The recursive unsolvability of the decision problem for the class of defnite formulas*. J. ACM 16(2) (1969) 324-327.

[Hir91] J. Hirschfeld, *Safe Queries in Relational Database with Functions*, in the Proceedings of the 5th Workshop on Computer Science Logic (Berne, 1991), pp.173-183, LNCS 626, Springer Verlag 1992.

[Ki88] M. Kiffer, *On Safety Domain independence and capturability of database queries*, Proc. International Conference on database and knowledge bases. Jerusalem, Israel (1988).

[Mal71] A.I. Mal'cev, *Axiomatizable classes of locally free algebras of various types*. In **The Metamathematics of Algebraic Systems. Collected Papers. 1936-1967**, pp. 262-289, North-Holland, 1971.

[Ni82] J.M. Nicolas *Logic for Improving Integrity Checking in Relational Data Bases*, Acta Informatica, Vol. 18 (1982), pp. 227-253.

[To87] R.W. Topor, *Domain independence formulas and databases.* Theoretical Computer Science 52 (1987) 281-306.

[Tra63] B.A. Trahktenbrot, *Impossibility of an Algorithm for the Decision Problem in Finite Classes*, American Mathematical Scoeity Transection, Series 2, Vol. 3 (1963), pp. 1-5.

[TS88] R.W. Topor and E.A. Sonenberg, *On domain independent databases*, In **Foundations of deductive databases and logic programming**, J. Minker (ed.), Morgan-Kaufmann, Los Altos, CA (1988) 217-240.

[Ull89] J.D. Ullman, **Principles of database and knowledge-base systems** (1989), Computer Science Press.