Computing with Infinitary Logic

Serge Abiteboul
INRIA, BP 105
78153 Le Chesnay CEDEX
France
email: abitebou@inria.inria.fr

Moshe Y. Vardi
IBM Almaden Research Center
San Jose, CA 95120-6099
USA
email: vardi@almaden.ibm.com

Victor Vianu
CSE C-0114 UC San Diego
La Jolla, CA 92093
USA
email: vianu@cs.ucsd.edu

November 10, 1993

Abstract

Most recursive extensions of relational calculus converge around two central classes of queries: *fixpoint* and *while*. Infinitary logic (with finitely many variables) is a very powerful extension of these languages which provides an elegant unifying formalism for a wide variety of query languages. However, neither the syntax nor the semantics of infinitary logic are effective, and its connection to practical query languages has been largely unexplored. We relate infinitary logic to another powerful extension of *fixpoint* and *while*, called *relational machine*, which highlights the computational style of these languages. Relational machines capture the kind of computation occurring when a query language is embedded in a host programming language, as in C+SQL. The main result of this paper is that relational machines correspond to the natural effective fragment of infinitary logic. Other well-known query languages are related to infinitary logic using syntactic restrictions formulated in language-theoretic terms. For example, it is shown that *while* corresponds to infinitary logic formulas which can be described by a regular language. As a side effect to these results, we obtain interesting normal forms for infinitary logic formulas.

1 Introduction

Recently, there has been considerable interaction between database theory and finite-model theory [F90]. A particularly useful formalism developed in finite-model theory is *infinitary logic*, an extension of first-order logic with infinite conjunctions and disjunctions [BF85]. Infinitary logic with finitely many variables (denoted $L^{\omega}_{\infty\omega}$) (see [Ba77]) provides an elegant framework for studying important phenomena such as 0-1 laws [KV90b] and expressive power of query languages [ACY91, KV90a].

While infinitary logic is an elegant theoretical tool, infinitary logic formulas have a non-effective syntax, and can define non-computable queries. Thus, infinitary logic is not practical in the context of databases. Nevertheless, most query languages are subsumed by the infinitary logic $L^{\omega}_{\infty\omega}$, which thus provides a useful unifying formalism for such languages. Indeed, $L^{\omega}_{\infty\omega}$ is not as exotic as it may sound; programs with recursion usually "compute" infinitary logic formulas. Consider for instance the Datalog program

$$\begin{array}{lcl} T(x,y) & \leftarrow & E(x,y) \\ T(x,y) & \leftarrow & E(x,z), T(z,y) \end{array}$$

The following equivalent infinitary logic formula is obtained by unfolding the computation of the above Datalog program:

$$\varphi = E(x,y) \vee (\exists z_1)(E(x,z_1) \wedge E(z_1,y)) \vee (\exists z_1,z_2)(E(x,z_1) \wedge E(z_1,z_2) \wedge E(z_2,y)) \vee \ldots$$

(As shown in Section 4, this can also be written more economically as a $L^{\omega}_{\infty\omega}$-formula, using just 3 variables which occur repeatedly.)

The purpose of this paper is to understand the connection between various query languages and infinitary logic, focusing primarily on $L^{\omega}_{\infty\omega}$, the infinitary logic with finitely many variables. The main result is to identify a natural *effective* fragment of infinitary logic, and relate it to known computational models for queries. We also show that well-known restricted database languages correspond to syntactic restrictions of the infinitary logics, described in language-theoretic terms.

To formalize the natural effective fragment of $L^{\omega}_{\infty\omega}$, we define syntactic and semantic notions of recursively enumerable (r.e.) formula. The semantic notion of r.e. requires the set of models of a sentence to be r.e.; the syntactic notion is based on the enumeration of the (syntactic) components of the sentence. The main result establishes an intimate connection between the effective fragments of $L^{\omega}_{\infty\omega}$ and a computing device which subsumes most database query languages. The device, called *typed relational machine*[1] (or simply *relational machine*), consists of a Turing Machine interacting with a *relational store* via fixed FO queries. This model is similar in spirit to the computation carried out in practical languages such as C+SQL, where FO queries are embedded in a host programming language. The connection with relational machines yields an interesting normal form for formulas in effective $L^{\omega}_{\infty\omega}$. This can be used to prove the robustness of the relational machine model; we can allow significantly more flexibility in the interface between the host programming language and the database, e.g., by including dynamically generated queries, and addressable relations, without increasing the expressive power of the model.

[1]The relational machine was introduced in [AV91] as GM^{loose} (Loosely-coupled Generic Machine). The term *relational machine* was first used in [AVV92].

Finally, we examine the connection between infinitary logic and well-known restricted query languages like *fixpoint* and *while* [CH82], the most well-studied extensions of FO, which motivated defining both infinitary logic and relational machines. We describe syntactic restrictions of infinitary logic corresponding to such languages, by characterizing the "shape" of allowed formulas. This is done in language-theoretic terms. For example, we show that *while* corresponds to infinitary logic formulas whose shape can be described by a regular language.

The paper is organized as follows. After a preliminary section, we define the effective fragments of infinitary logics and relate them to relational machines. The connection with restricted query languages, yielding characterizations in language-theoretic terms, is considered in Section 4.

2 Preliminaries

We assume basic knowledge of databases [K90, U88] and formal language theory [HU79]. In databases, only finite structures are considered. Most traditional query languages are based on first-order logic without function symbols (here FO). The simplicity of Codd's algebraization of FO and the fact that FO is in (uniform) AC_0 [BIS90] (and, thus, in a reasonable sense, takes constant parallel time) explain the appeal of FO as a query language. In this section, we define the infinitary logics $L_{\infty\omega}$ and $L^{\omega}_{\infty\omega}$ [Ba77] and briefly present the database languages and computational devices that are considered in the paper, namely, *fixpoint*, *while*, and relational machines.

Fixpoint and While

Most of the extensions of FO with recursion that have been proposed converge towards two classes of queries, *fixpoint* and *while* [CH82]. Although widely varying paradigms can be used to extend FO (e.g. procedural, logic, or logic programming), most languages based on the various paradigms are equivalent to either the *fixpoint* or the *while* queries.

The *fixpoint queries* (*fixpoint*) [CH82] are constructed using the first-order constructors as in FO together with a fixpoint operator (μ). The fixpoint operator binds a predicate symbol T that is free and that appears only positively (i.e., under an even number of negations) in the formula. The semantics is given by the least fixpoint of the formula, and convergence is guaranteed in polynomial time. For instance, the transitive closure of relation E is given by the *fixpoint* query $\mu_T \varphi(x, y, T)$ for

$$\varphi = E(x, y) \vee \exists z(E(x, z) \wedge T(z, y)).$$

Fixpoint expresses exactly PTIME on ordered databases [I86, V82]. It cannot, however, express the evenness[2] query.

The *while* language was originally introduced in [CH82] in a procedural form. FO is extended with (i) sorted relational variables ($X, Y, \ldots$), (ii) assignment of FO queries to variables, and (iii) a *while* construct allowing to iterate a program while some condition (e.g., $X = \emptyset$) holds. An alternative definition of *while* based on *partial fixpoint logic* is

[2]The evenness query on a set S is the query *even(S)*= **true** iff $|S|$ is even.

proposed in [AV88]. *While* expresses PSPACE on ordered databases [V82], but, like *fixpoint*, cannot express the evenness query on an unordered set.

Infinitary Logic

As stated above, *while* cannot compute the evenness query on a set. An elegant proof uses the fact that all properties expressible by *while* have a *0-1 law*, i.e. the asymptotic probability that the property holds exists and is either 0 or 1 [KV87]. It is of interest whether the 0-1 law holds for more powerful languages. Infinitary logic with finitely many variables is a very powerful extension of *while* that still has a 0-1 law [KV90b]. This provides insight into the factors that limit expressive power in query languages. Beyond this aspect, $L^{\omega}_{\infty\omega}$ provides an elegant unifying formalism for a wide variety of query languages [ACY91, KV90b].

We first define unrestricted infinitary logic. $L_{\infty\omega}$ is first-order logic (FO) extended by allowing disjunctions and conjunctions of infinite sets of formulas.

Definition: The set of *infinitary formulas* over a database schema[3] σ is the smallest collection $L_{\infty\omega}$ of expressions such that

1. it contains all FO-formulas over σ;
2. if φ is an infinitary formula and x a variable, then $\neg(\varphi), \exists x(\varphi), \forall x(\varphi)$, are also infinitary formulas; and
3. if A is a set of infinitary formulas, then $\bigvee A$ and $\bigwedge A$ are also infinitary formulas.

The semantics is the natural extension of FO semantics. We are particularly interested by infinitary formulas in which the number of variables is finite. (Each variable may occur an unbounded number of times in the formula.) This logic has a 0-1 law, while the unrestricted logic can define all isomorphically closed classes of finite structures and, consequently, is of little interest in the context of finite structures.

Definition: Let k be a positive integer. The set of *infinitary formulas with k variables* over σ is the collection $L^k_{\infty\omega}$ of $L_{\infty\omega}$-formulas with at most k variables (with possibly infinitely many occurrences) and

$$L^{\omega}_{\infty\omega} = \bigcup_{k=1}^{\infty} L^k_{\infty\omega}.$$

The collection of FO-formulas with at most k variables is denoted FO^k.

Relational Machines

The *fixpoint* and *while* queries can be viewed as programming languages with simple control interacting with the databases via a fixed set of FO queries. A natural extension of this paradigm is to consider a computationally complete computing device interacting with the database via given FO queries. This is in the spirit of SQL queries embedded in a full

[3] A database schema is a finite set of relation names and their arities.

programming language such as C. It is formally captured by a computational model called "relational machine".

A *typed relational machine* (or simply *relational machine*) is a Turing Machine (TM) augmented with a finite set of fixed-arity relations forming a *relational store*. The relational store can be viewed as associative access storage supporting the generic portion of the computation, while standard computation is carried out on the tape. Designated relations contain initially the input, and others hold the output at the end of the computation. In a transition, the relational store can be modified through a first-order (FO) query (equivalently, through relational algebraic operations). The *arity* of a relational machine is the maximum number of variables used in its FO queries.

The connection between relational machines and *while* is emphasized by a result showing that relational machines are equivalent to *while* extended with integer variables and arithmetic [AV91]. The resulting language is called *while*$^+$. Besides the relational variables, integer variables are used with the operations *increment, decrement* and a test for *zero*. An extension *while*$^{++}$, obtained by allowing mixing integers with the data in relations, remains equivalent to *while*$^+$ [AV91].

Although the TM component of relational machines provides full computational power, relational machines do *not* express all computable queries. For instance, they cannot compute the evenness query. One can obtain a complete device by allowing untyped relations in the relational store, whose arity can vary throughout the computation. The resulting device is called *untyped relational machine*. This is in the spirit of the first query-complete language proposed in [CH80]. An equivalent *typed* version of these machines has been defined in [AV91] under the name *Generic Machine (GM)*. A generic machine compensates for the typedness by allowing the transfer of data between the relational store and the Turing tape. This also involves parallel computation provided by spawning several "unit GMs" working synchronously.

A normal form

We informally describe a powerful normal form for relational machines, shown in [AV91], which provides a key technical tool. It states that any relational machine computation on an unordered input can be reduced to a computation on an ordered input via a *fixpoint* query. More precisely, for each relational machine program p there exist a *fixpoint* program f and another relational machine p' such that

- $p \equiv fp'$;
- the output of f is an ordered instance;
- p' is a relational machine program operating on the ordered instance output by f.

3 Effective Infinitary Logic and Relational Machines

In this section, we formalize the notion of effective fragment of $L^{\omega}_{\infty\omega}$ and show the correspondence with relational machines.

As noted, neither the syntax nor the semantics of $L^{\omega}_{\infty\omega}$ are effective. To isolate natural effective fragments of $L^{\omega}_{\infty\omega}$, we consider both the syntactic and the semantic aspects. This leads to two notions of r.e. $L^{\omega}_{\infty\omega}$-formula:

- semantic: the set of (finite) models of the formula is r.e.; and
- syntactic: one can effectively enumerate the "components" of the formula.

As we shall see, although the two approaches do not lead to the same classes of formulas (nor the same sets of models), they are closely related.

Consider first the semantic approach.

Definition: A set S of finite structures over some schema σ is *recursively enumerable* (r.e.) if there exists a recursive enumeration $I_1, ..., I_i, ...$ of structures over σ such that I is in S iff some structure isomorphic to I belongs to the enumeration.

The semantic notion of effective $L^{\omega}_{\infty\omega}$-formula requires that the set of models of the formula be r.e.

For the syntactic approach, consider the syntax tree of a given $L^{\omega}_{\infty\omega}$-formula φ. By definition, the depth of this tree is bounded by some integer d. Suppose that each node in the tree has countably many children. Then each position in the tree is identified by a sequence $i_1...i_k$ of integers, $1 \leq k \leq d$. The notion of recursive enumeration of the components of the formula is therefore captured by:

Definition: A formula φ in $L^{\omega}_{\infty\omega}$ is *r.e.* if:

1. each of its disjunctions and conjunctions is countable; and
2. there exists a computable function which, given a sequence of integers $i_1...i_k$, $1 \leq k \leq d$ (where d is the depth of the formula), returns the symbol in the position denoted by $i_1...i_k$ in the syntax tree of φ.

Remark: The above definition is the analogue of the definition of the Church-Kleene fragment $L^{CK}_{\omega_1,\omega}$, which is the syntactically effective fragment of $L_{\omega_1,\omega}$; see [Ba75].

It turns out that syntactic effectiveness does not guarantee semantic effectiveness. Indeed, we will see that the r.e. formulas yield the analog of the arithmetic hierarchy defined using relational machines. This is closely related to the standard arithmetic hierarchy.

We next characterize the $L^{\omega}_{\infty\omega}$-formulas whose set of models is r.e. The main result shows that these are equivalent to relational machines. This also yields a syntactic normal form for such formulas.

Theorem 3.1 Let S be a set of finite structures over some fixed schema. For each k greater than the maximum arity in S, the following are equivalent:

(i) S is accepted by some relational machine of arity k;

(ii) $S = models(\varphi)$ for some $\varphi \in L^{k}_{\infty\omega}$ and S is r.e.;

(iii) $S = models(\bigvee A)$, for some recursive set A of FO^k sentences.

Proof: (sketch) (i) $\Rightarrow$ (iii). Each accepting computation of a relational machine can be described by an FO^k-formula. We can assume without loss of generality that the machine applies a first-order transformation to the relations in its store whenever it makes a transition. Thus, the above formula is larger than the length of the computation. An input is accepted iff it is accepted by some accepting computation, i.e. it satisfies $\bigvee A$ where A is the set of formulas describing accepting computations. Finally, note that A is recursive.

(iii) $\Rightarrow$ (ii) Suppose $S = models(\bigvee A)$, for some recursive set A of FO^k sentences. Let $\varphi = \bigvee A$. Then φ is in $L^k_{\infty\omega}$. It remains to show that S is r.e. To enumerate S, one can enumerate (up to isomorphism) the pairs $[I, \psi]$ where I is an instance and ψ is in A, and output I if $I \models \psi$.

(ii) $\Rightarrow$ (i). Suppose that $S = models(\varphi)$ is r.e., where $\varphi \in L^k_{\infty\omega}$. Since φ is in $L^k_{\infty\omega}$, we can use the normal form theorem [AV92], i.e., rewrite φ as $f \circ \psi$ where:

- f is a fixpoint query whose output is an ordered structure and
- ψ is obtained from φ by replacing each FO^k-formula by a formula working on the ordered output of f.

Since relational machines subsume *fixpoint*, f is computable by such a machine. Next, note that $f(models(\varphi))$ is r.e. (since $models(\varphi)$ is r.e.). Because relational machines are complete on ordered structures, it follows that there is a relational machine simulating ψ which accepts $f(models(\varphi))$. By composing the machine for f and that for $f(models(\varphi))$, we obtain a relational machine accepting $models(\varphi)$. $\Box$

We note that the normal form provided by (iii) above can be viewed as the effective counterpart of a similar result shown for the full $L^k_{\infty\omega}$ in [KV92]. It is also shown in [KV92] that every formula of $L^k_{\infty\omega}$ is equivalent to a countable disjunction of FO^l-formulas, for some $l \geq k$. Furthermore, it follows from [DLW91] that this holds also for $l = k$.

To complete the picture, we next make a connection between the (syntactically) r.e. $L^\omega_{\infty\omega}$-formulas and relational machines. We show that the formulas correspond to the arithmetic hierarchy based on relational machines.

The arithmetic hierarchy based on relational machines is defined in analogy to the classical arithmetic hierarchy. More precisely, add oracles for sets of structures accepted by relational machines to obtain the first level of the hierarchy; next, add oracles on the set of structures that can thereby be accepted; and so on. Let Σ^{rel} denote the collection of sets of structures in this hierarchy.

We prove:

Proposition 3.2 $S = models(\varphi)$ for some r.e. formula φ in $L^\omega_{\infty\omega}$ iff S is in Σ^{rel}. $\Box$

It is easily seen that Σ^{rel} coincides on ordered inputs with the classical arithmetic hierarchy. Thus, r.e. $L^\omega_{\infty\omega}$-formulas can define non-computable queries.

The terminating case

Not surprisingly, terminating machines (i.e., machines that terminate on all inputs) are related to recursiveness of models. This is shown in the next result.

Theorem 3.3 Let S be a set of finite structures over some schema σ. The following are equivalent for each k greater that the maximum arity in a relation in σ:

(i) S is accepted by some terminating relational machine of arity k;

(ii) $S = models(\varphi)$ for some φ in $L^k_{\infty\omega}$ and S is recursive; and

(iii) $S = models(\bigvee A)$ and $S = models(\bigwedge B)$ for some recursive sets A, B of formulas in FO^k.

Theorems 3.1 and 3.3 provide normal forms for $L^k_{\infty\omega}$-formulas whose sets of models are r.e. or recursive.

Corollary 3.4 1. Each $L^k_{\infty\omega}$-formula φ such that $models(\varphi)$ is r.e. is equivalent to some formula $\bigvee A$ where A is a recursive set of FO^k-formulas.

2. For each $L^k_{\infty\omega}$-formula φ such that $models(\varphi)$ is recursive, φ is equivalent to some formulas $\bigvee A, \bigwedge B$, where A and B are recursive sets of FO^k

Remark: It is interesting to note that the model of relational machines (or equivalently, the semantically r.e. fragment of $L^\omega_{\infty\omega}$) is quite robust. It is possible to extend relational machines with more flexible interfaces to the relational store without increasing the expressive power of the model.

One such extension allows to mix relational calculations with integer arithmetics. That is, one can allow the use of integer variables (in the style of *while*$^+$) with increment, decrement, and a test for zero; or more generally, the use of integer arithmetic. Integers can even be stored in database relations, in the style of *while*$^{++}$. This does not increase the expressive power of the model.

Another extension allows dynamic generation of queries. That is, one can add a "query tape" to the machine. The machine can generate dynamically (encodings of) FO queries using up to k variables on the query tape. The machine can request at any time the application of the query on the tape against the relational store, and use it either as a test in the control or to assign the result to a relation in the store. This also does not increase the expressive power of the model.

Finally, a third extension allows addressable relations, i.e. the relational store has a variable, unbounded number of relations of bounded arity. This extension does not affect expressive power either. $\square$

4 Languages-based Restrictions of $L^\omega_{\infty\omega}$

In this section, we provide a language-theoretic characterization of *fixpoint* and *while* queries as syntactic fragments of infinitary logic.[4] The crux of the approach is to characterize the shape of desired formulas in terms of languages (such as regular languages).

[4]We note that Shemesh and Francez [SF88] attempted to provide a language-theoretic characterization of Datalog queries. Their approach, which is quite different than ours, is in terms of what they call "Datalog automata" and it does not completely characterize Datalog queries.

To clarify the idea, we begin with an example. Consider a graph E and the formula stating that there is a path from x to y. It is expressed by the following infinitary formula:

$$\varphi_1 = E(x,y) \vee (\exists x_1)(E(x,x_1) \wedge E(x_1,y)) \vee (\exists x_1,x_2)(E(x,x_1) \wedge E(x_1,x_2) \wedge E(x_2,y)) \vee ...$$

In fact, it turns out (cf. [KV90b]) that this can be rewritten using just 3 variables, as the formula $\varphi_2 = \bigvee_{i=1}^{\infty} p_i$ where:

$$p_1(x,y) = false(x,y)$$
$$p_n(x,y) = E(x,y) \vee p_{n-1}(x,y) \vee (\exists z)[(E(x,z) \wedge (\exists x)(x = z \wedge p_{n-1}(x,y)))].$$

with $false(x,y) = x = y \wedge x \neq y$. In the first formulation, φ_1 is in $L_{\infty\omega}$ whereas in the second, φ_2 is in $L^3_{\infty\omega}$.

Consider the formula φ_2 for the connectivity of two vertices, and the language $L = \{p_i \mid i \geq 1\}$. Note that L is a language over a finite alphabet. We will use a formulation that will make the connection to standard language formalism more obvious. Consider

$$\varphi(P(x,y)) = E(x,y) \vee P(x,y) \vee (\exists z)[(E(x,z) \wedge (\exists x)(x = z \wedge P(x,y)))].$$

Observe that φ_2 can be written as:

$$\varphi(false(x,y)) \vee \varphi(\varphi(false(x,y))) \vee ... \vee \varphi^i(false(x,y)) \vee ...$$

using composition of formulas. The connection with languages is based on an analogy between composition of formulas and concatenation of symbols. Indeed, if composition is denoted by concatenation, the formula φ_2 can be written as $\bigvee \varphi^+$, where $\varphi^+ = \{\varphi^i(false) \mid i \geq 1\}$; this highlights clearly the analogy with regular languages.

We now present more formally the notion of composition of formulas. When composing a formula φ with a formula ψ, ψ is substituted in place of some relation occurring in φ. Thus, we assume that each formula used in such compositions has a specified relation symbol which is to be used in compositions. For example, for φ above the relation symbol is P. We shall denote this by $\varphi(P)$ and will call P the *carrier* of compositions for φ. Clearly, only formulas ψ with the same number of free variables as the arity of the carrier P can be composed with $\varphi(P)$. The carrier is only used to define compositions, just like the carrier in a *fixpoint* formula is used in the inductive definition of a relation starting from $false(x,y)$. In both cases, the carrier is *not* part of the database schema. However, $\varphi(false)$ contains only database relations and so can be evaluated against the database.

We now develop more formally the analogy between languages and formulas. Let σ be a database schema and Σ a finite set of FO^k-formulas each of which is of the form $\varphi(P)$ with $P \notin \sigma$ and φ is over $\sigma \cup \{P\}$. Let $\varphi(P)$ and $\psi(Q)$ be in Σ such that the number of free variables of $\psi(Q)$ equals the arity of P; then the composition $(\psi\varphi)(Q)$ of $\psi(Q)$ and $\varphi(P)$ is obtained by replacing every atom $P(\mathbf{t})$ occurring in $\varphi(P)$ by $\psi(Q)(\mathbf{t})$. Note that the carrier in $\psi\varphi$ is now Q. Let Σ^* denote the closure of Σ under composition of formulas.

Let Γ be a finite alphabet and h a mapping ¿from Γ to Σ. The mapping h is extended to a partially defined homomorphism from Γ^* to Σ^* by $h(ab) = h(a)h(b)$ if the composition of $h(a)$ with $h(b)$ is well defined. This can be further extended from *languages* over Γ to $L^k_{\infty\omega}$-formulas by

$$h(L) = \bigvee\{h(w)(false) \mid w \in L \text{ and } h(w) \text{ is well defined}\}.$$

Fact: It is easily verified that, with Γ, h and Σ as above, the language

$$\{w \in \Gamma^* \mid h(w) \text{ is well defined}\}$$

is a regular language.

Thus, if L is a language, the subset of L consisting of the words w for which $h(w)$ is well defined is of the form $L \cap R$, where R is regular. Recall that all standard classes of formal languages are closed under intersection with regular sets.

By abuse of notation, we use the formula $h(a)$ instead of a as a letter of the alphabet Γ. So, for instance, for two formulas φ and ψ, we will refer to the language $\varphi^*\psi$ meaning $h(a^*b)$ for some alphabet $\{a,b\}$ and the homomorphism h such that $h(a) = \varphi$ and $h(b) = \psi$.

Let $\mathcal{L}$ be a set of languages. An $L^k_{\infty\omega}$-formula ξ is said to be *$\mathcal{L}$-definable* (or simply $\mathcal{L}$) if it is $h(L)$ for some L in $\mathcal{L}$ and some h. One can thus define the regular formulas, the context-free, or context-sensitive formulas.

We first consider the *while* language.

Theorem 4.1 A query is in *while* iff it is definable by a regular $L^\omega_{\infty\omega}$-formula.

Proof: (sketch) By the equivalence of *while* with the partial fixpoint logic FO+PFP [AV88] and by the collapse of the FO+PFP hierarchy, each *while* query can be written as $\bigvee \varphi^*\psi$ for some FO-formulas φ and ψ (in fact, ψ is a selection-projection). Thus, every query in *while* is definable by a regular $L^\omega_{\infty\omega}$-formula. Conversely, let ξ be a regular $L^\omega_{\infty\omega}$-formula. That is, ξ is $h(L)$ for some regular language L. Let M be a finite-state automaton that accepts L. A nondeterministic relational machine can compute the query ξ by guessing a path from a starting state to an accepting state in M and applying the formulas that correspond to the letters along that path. The machine does not use the tape at all. It is shown in [AVV92] that such nondeterministic relational machines can be simulated by *while* queries. □

The above proof provides a non-obvious normal form for all regular $L^\omega_{\infty\omega}$-formulas.

Corollary 4.2 Each regular $L^\omega_{\infty\omega}$-formula is equivalent to a regular $L^\omega_{\infty\omega}$-formula of the form $\varphi^*\psi$ (where ψ is a selection-projection).

We next consider the *fixpoint* queries. The language-theoretic characterization of fixpoint queries is in terms of one-letter languages and monotonic formulas. A language L is a *one-letter end-marked language* if it is over a two-letter alphabet $\{a,b\}$ and it is a subset of the language a^*b. Intuitively, we can think of L as a language over the one-letter alphabet $\{a\}$ with the end-marker b. A formula χ of $L^\omega_{\infty\omega}$ is a *monotonic one-letter end-marked formula* if it definable in terms of a one-letter end-marked language over a set Σ of monotonic formulas.

Theorem 4.3 A query is in *fixpoint* iff it is definable by a regular monotonic one-letter end-marked $L^\omega_{\infty\omega}$-formula. □

It is interesting to note that *while* can also be characterized in terms of one-letter end-marked formulas (without the monotonicity requirement). On the other hand, the restriction to one-letter end-marked formulas in Theorem 4.3 is crucial, since otherwise nondeterminism would be required by a relational machine that is trying to simulate the formula. Such nondeterministic computations are not known to be in *fixpoint*. In the full paper we will show that regular monotonic formulas correspond to the *nondeterministic inflationary fixpoint logic* of [AVV92].

The results in this section raise the natural issue of the characterization of other classes of $L^{\omega}_{\infty\omega}$ formulas, such as the context-free or context-sensitive $L^{\omega}_{\infty\omega}$-formulas. Such characterizations remain an open problem.

References

[ACY91] Afrati, F., S.S. Cosmadakis, M. Yannakakis, On Datalog vs. polynomial time, *Proc. 10th ACM Symp. on Principles of Database Systems*, 1991.

[AV88] Abiteboul, S., V. Vianu, Datalog extensions for database updates and queries, *J. Computer and System Sciences* 43:1(1991).

[AV91] Abiteboul, S., V. Vianu, Generic computation and its complexity, *Proc. 23rd ACM Symposium on Theory of Computing*, 1991.

[AV92] Abiteboul, S., V. Vianu, Computing with first-order logic, to appear in *J. Computer and System Sciences.*

[AVV92] Abiteboul, S., M.Y. Vardi, V. Vianu, Fixpoint logics, relational machines, and computational complexity, to appear in *Structures in Complexity Theory, Proc. 7th IEEE Conf.*.

[Ba75] Barwise, J., *Admissible Sets and Structures*, Springer-Verlag, 1975.

[Ba77] Barwise, J., On Moschovakis closure ordinals, *J. Symbolic Logic*, 42 (1977), pp. 292–296.

[BF85] Barwise, J., S. Feferman (eds.), *Model-Theoretic Logics*, Springer-Verlag, 1985.

[BIS90] D. A. M. Barrington, N. Immerman, and H. Straubing, On uniformity within NC^1, *J. Computer and System Sciences* 41(1990), 274–306.

[CH80] Chandra, A.K., D. Harel, Computable Queries for Relational Databases, *J. Computer and System Sciences* 21:2 (1980), 156–178.

[CH82] Chandra, A.K., D. Harel, Structure and Complexity of Relational Queries, *J. Computer and System Sciences* 25:1 (1982), 99–128.

[DLW91] Dawar, A., S. Lindell and S. Weinstein, *Infinitary logic and inductive definability over finite structures*, Technical Report, Univ. of Pennsylvania, 1991.

[F90] Fagin R., Finite-Model Theory—a Personal Perspective, Proc. 3rd Int'l. Conf. on Database Theory, Springer-Verlag, Lecture Notes in Computer Science 470, 1990, pp. 3–24, to appear in *Theoretical Computer Science.*

[HU79] Hopcroft J.E., J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[I86] Immerman N., Relational queries computable in polynomial time, *Information and Control* 68 (1986), pp. 86–104.

[K90] Kanellakis, P.C., Elements of Relational Database Theory, *Handbook of Theoretical Computer Science* (J. van Leeuwen, A.R. Meyer, N. Nivat, M.S. Paterson, and D. Perrin, eds.), Vol. B, Chapter 17, North-Holland, 1990.

[KV87] Kolaitis, P., M.Y. Vardi, The decision problem for the probabilities of higher-order properties, *Proc. 19th ACM Symp. on Theory of Computing*, 1987, pp. 425–435.

[KV90a] Kolaitis, P., M.Y. Vardi, On the expressive power of Datalog: tools and a case study, *Proc. 9th ACM Symp. on Principles of Database Systems*, 1990, pp. 61–71, to appear in J. Computer and System Sciences.

[KV90b] Kolaitis, P., M.Y. Vardi, 0-1 laws for infinitary logic, *Proc. 5th IEEE Symp. on Logic in Computer Science*, 1990, pp. 156–167. To appear in *Information and Computation.*

[KV92] Kolaitis, P., M.Y. Vardi, Fixpoint vs. infinitary logic in finite-model theory, to appear in *Proc. 7th IEEE Symp. on Logic in Computer Science*, 1992.

[SF88] Shemesh Y., N. Francez, *Finite-state Datalog automata and relational languages*, unpublished manuscript, 1988.

[U88] Ullman, J.D., *Principles of Database and Knowledge Base Systems*, Computer Science Press, 1988.

[V82] Vardi, M.Y., The complexity of relational query languages, *Proc. 14th ACM Symp. on Theory of Computing*, 1982, pp. 137–146.